EVAT DATA MODEL DEVELOPMENT

Leverages a EV reference Charging Session Transactions weekly/annual charging patterns.

Localises the charging Session transaction to Melbourne Victoris.

Update the charging session transactions with modern day EV battery capacities.

Incorporates Victoria Wholesale Electricity Prices

Creation of Database Table CSV Files

Lists additional inter-table Entity relationships required to construct a Database

Importation of EV REFERENCE Charging Session Transactions& Data Analysis

Data Preprocessing

```
#import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
#load dataset
df= pd.read_csv('station_data_dataverse.csv')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3395 entries, 0 to 3394
Data columns (total 24 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   sessionId       3395 non-null   int64  
 1   kwhTotal        3395 non-null   float64
 2   dollars         3395 non-null   float64
 3   created         3395 non-null   object 
 4   ended           3395 non-null   object 
 5   startTime       3395 non-null   int64  
 6   endTime         3395 non-null   int64  
 7   chargeTimeHrs   3395 non-null   float64
 8   weekday         3395 non-null   object 
 9   platform        3395 non-null   object 
 10  distance        2330 non-null   float64
 11  userId          3395 non-null   int64  
 12  stationId       3395 non-null   int64  
 13  locationId      3395 non-null   int64  
 14  managerVehicle  3395 non-null   int64  
 15  facilityType    3395 non-null   int64  
 16  Mon             3395 non-null   int64  
 17  Tues            3395 non-null   int64  
 18  Wed             3395 non-null   int64  
 19  Thurs           3395 non-null   int64  
 20  Fri             3395 non-null   int64  
 21  Sat             3395 non-null   int64  
 22  Sun             3395 non-null   int64  
 23  reportedZip     3395 non-null   int64  
dtypes: float64(4), int64(16), object(4)
memory usage: 636.7+ KB
```

```
#display columns
df.columns
```

```
Index(['sessionId', 'kwhTotal', 'dollars', 'created', 'ended', 'startTime',
       'endTime', 'chargeTimeHrs', 'weekday', 'platform', 'distance', 'userId',
       'stationId', 'locationId', 'managerVehicle', 'facilityType', 'Mon',
       'Tues', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun', 'reportedZip'],
      dtype='object')
```

```
#calculate sum of null values
df.isnull().sum()
```

	0
sessionId	0
kwhTotal	0
dollars	0
created	0
ended	0
startTime	0
endTime	0
chargeTimeHrs	0
weekday	0
platform	0
distance	1065
userId	0
stationId	0
locationId	0
managerVehicle	0
facilityType	0
Mon	0
Tues	0
Wed	0
Thurs	0
Fri	0
Sat	0
Sun	0
reportedZip	0

dtype: int64

```
#display all columns
pd.set_option('display.max_columns', None)
```

```
df.head(2)
```

	sessionId	kwhTotal	dollars	created	ended	startTime	endTime	chargeTimeHrs	weekday	platform	distance	user
0	4926737	23.68	0.5	0015-10-03 07:18:43	0015-10-03 11:25:07	7	11	4.106667	Sat	android	NaN	789081
1	3738844	22.14	0.0	0015-08-01 05:29:02	0015-08-01 09:00:08	5	9	3.518333	Sat	android	NaN	789081

Finding the number of Unique stations, Users, Charge Sessions

```
#number of duplicate entries
print('Duplicate entries found in this dataset = {}.' .format(df.duplicated().sum()))
```

```
Duplicate entries found in this dataset = 0.
```

```
# count numner of unique ussessionId in df
unique_session_count = df['sessionId'].nunique()
print(f"The number of unique session IDs is: {unique_session_count}")
```

```
The number of unique session IDs is: 3395
```

```
# count numner of unique userId in df
unique_user_count = df['userId'].nunique()
print(f"The number of unique user IDs is: {unique_user_count}")
```

```
The number of unique user IDs is: 85
```

```
# count numner of unique locationId in df
unique_location_count = df['locationId'].nunique()
print(f"The number of unique location IDs is: {unique_location_count}")
```

```
The number of unique location IDs is: 25
```

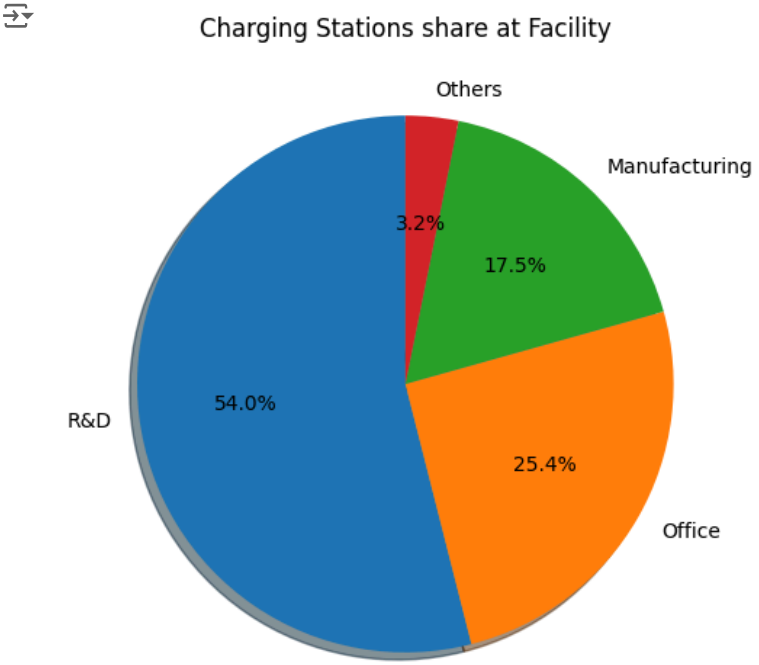
```
# count numner of unique stationId in df
unique_station_count = df['stationId'].nunique()
print(f"The number of unique station IDs is: {unique_station_count}")
```

```
The number of unique station IDs is: 105
```

Installed Charging Stations Share at the Facility

```
# create a new column and convert 1 to manufacturing, 2 to office, 3 to R&D and 4 to others in facilityType
df['facilityType'] = df['facilityType'].replace([1,2,3,4],['Manufacturing','Office','R&D','Others'])
```

```
#pie chart of facilityType
plt.figure(figsize=(10,6))
df['facilityType'].value_counts().plot(kind='pie', autopct='%1.1f%%', shadow=True, startangle=90)
plt.title('Charging Stations share at Facility')
plt.ylabel('')
plt.show()
```



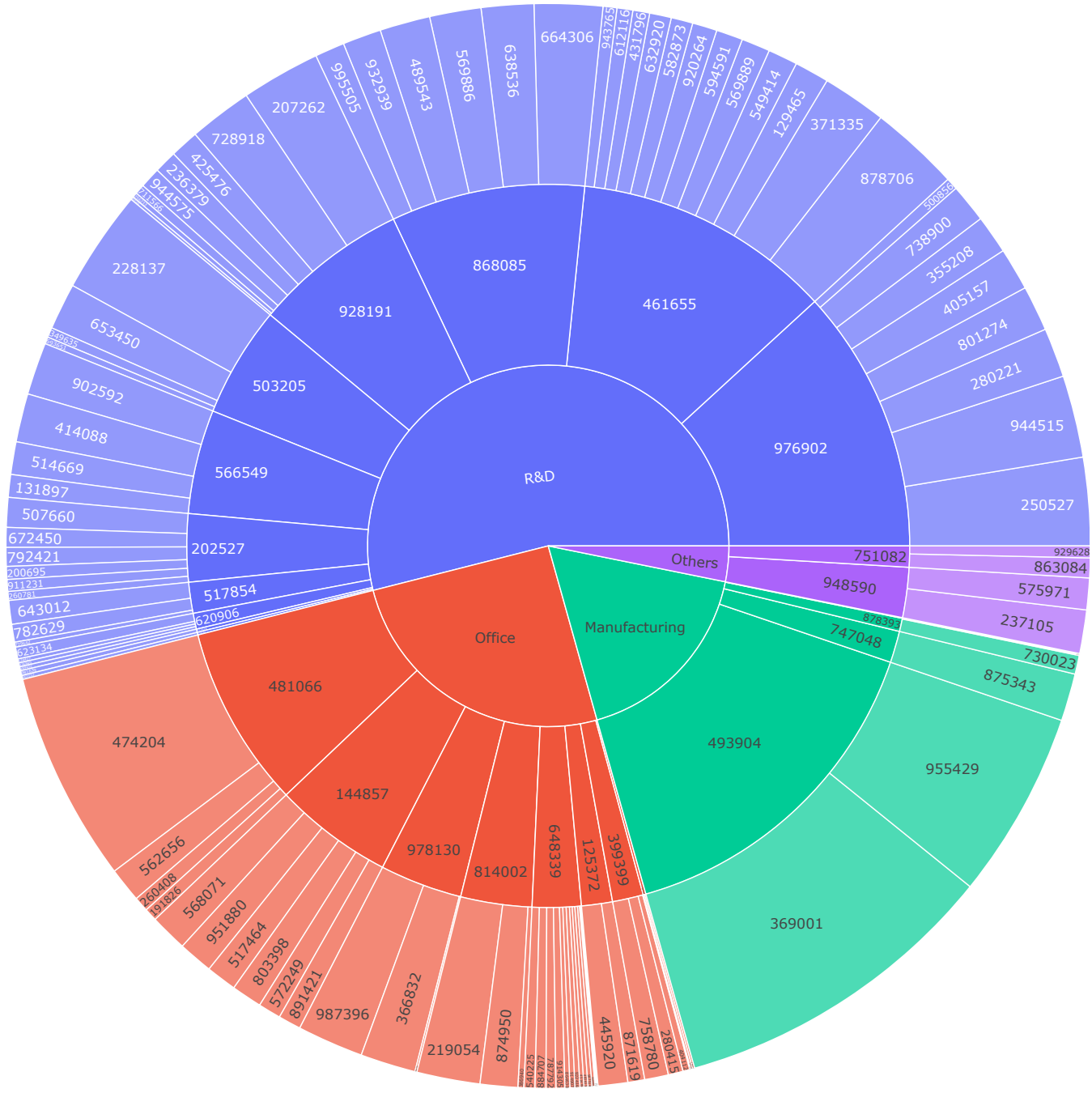

```
!pip install plotly.express
import plotly.express as px
```

```
Collecting plotly.express
  Downloading plotly_express-0.4.1-py2.py3-none-any.whl.metadata (1.7 kB)
Requirement already satisfied: pandas>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from plotly.express) (2.2.2)
Requirement already satisfied: plotly>=4.1.0 in /usr/local/lib/python3.10/dist-packages (from plotly.express) (5.24.1)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from plotly.express) (0.14
Requirement already satisfied: scipy>=0.18 in /usr/local/lib/python3.10/dist-packages (from plotly.express) (1.13.1)
Requirement already satisfied: patsy>=0.5 in /usr/local/lib/python3.10/dist-packages (from plotly.express) (1.0.1)
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.10/dist-packages (from plotly.express) (1.26.4)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.20.0->p
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.20.0->plotly.expr
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.20.0->plotly.ex
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly>=4.1.0->plotly.ex
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from plotly>=4.1.0->plotly.express)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas>
Downloading plotly_express-0.4.1-py2.py3-none-any.whl (2.9 kB)
Installing collected packages: plotly.express
Successfully installed plotly.express-0.4.1
```

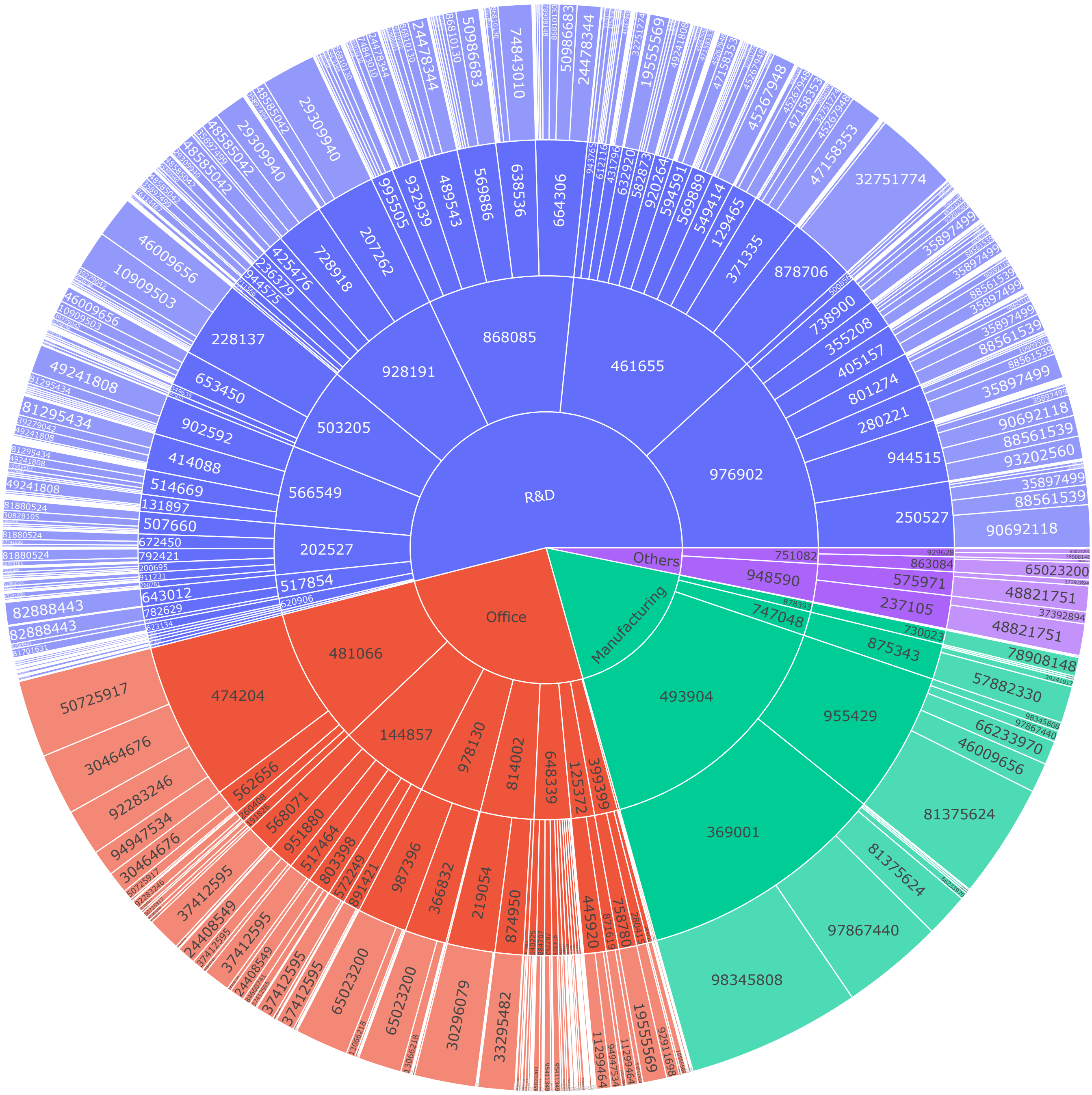
```
df.head(5)
```

	sessionId	kwhTotal	dollars	created	ended	startTime	endTime	chargeTimeHrs	weekday	platform	distance	user
0	4926737	23.68	0.5	0015-10-03 07:18:43	0015-10-03 11:25:07	7	11	4.106667	Sat	android	NaN	789081
1	3738844	22.14	0.0	0015-08-01 05:29:02	0015-08-01 09:00:08	5	9	3.518333	Sat	android	NaN	789081
2	2682332	22.07	0.0	0015-09-02 13:43:27	0015-09-02 17:38:10	13	17	3.911944	Wed	android	NaN	789081
3	9025610	22.03	0.0	0015-05-29 16:55:35	0015-05-29 20:54:06	16	20	3.975278	Fri	android	NaN	789081
4	4473237	21.20	0.0	0015-09-30 16:52:49	0015-09-30 20:42:06	16	20	3.821389	Wed	android	NaN	789081

```
#Plot Start Burxt diagram
fig = px.sunburst(
    df,
    path=['facilityType', 'locationId', 'stationId', ],  # Changed 'facilityTypen' to 'facilityType'
    height=1000
)

fig.update_layout(margin=dict(t=0, l=0, r=0, b=0))

fig.show()
```




```
#Plot Start Burxt diagram
fig = px.sunburst(
    df,
    path=['facilityType', 'locationId', 'stationId','userId' ],  # Changed 'facilityTypen' to 'facilityType'
    height=1000
)

fig.update_layout(margin=dict(t=0, l=0, r=0, b=0))

fig.show()
```




```
# prompt: group by facilityType and sum of kwhTotal

grouped_df = df.groupby('facilityType')['kwhTotal'].sum()
grouped_df
```

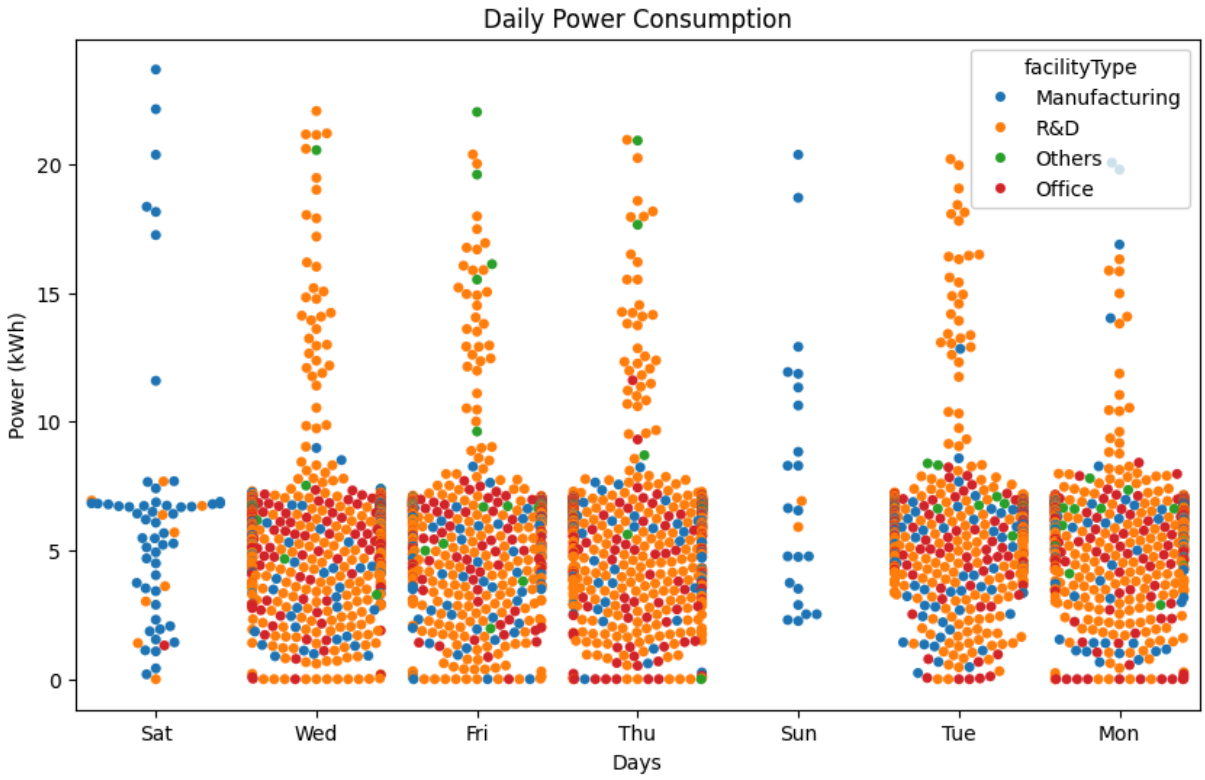
facilityType	kwhTotal
Manufacturing	3411.26
Office	4829.75
Others	779.44
R&D	10703.24

dtype: float64

Daily Power Consumption

```
#convert facilityType to Catagorical data
df['facilityType'] = df['facilityType'].replace([1,2,3,4],['Manufacturing','Office','R&D','Others'])

#visualize daily power consumption
plt.figure(figsize=(10,6))
sns.swarmplot(data=df, y='kwhTotal', x='weekday', hue='facilityType')
plt.title('Daily Power Consumption')
plt.xlabel('Days')
plt.ylabel('Power (kWh)')
plt.show()
```

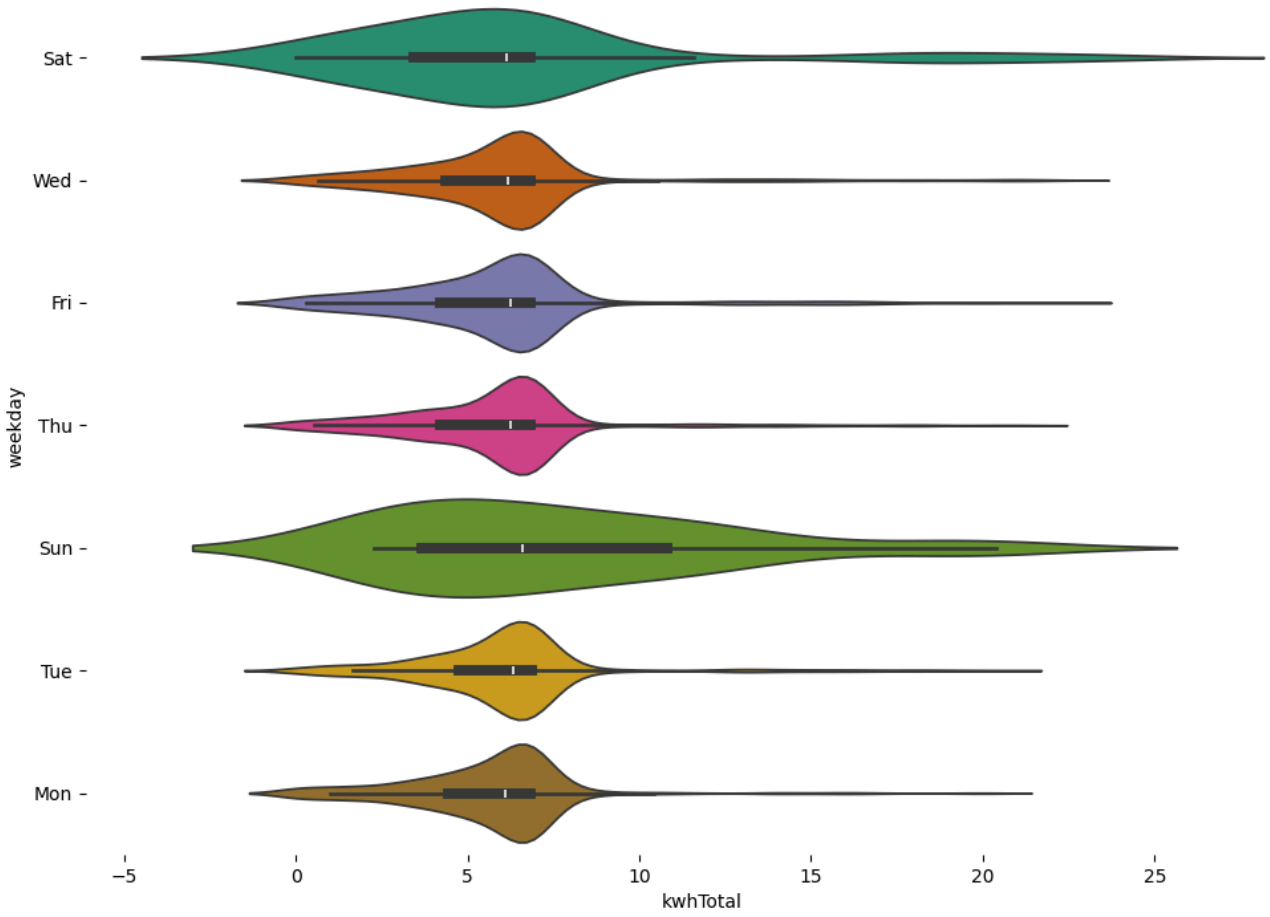
```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
60.6% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
53.9% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
60.7% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
58.4% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
57.6% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
6.5% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
62.0% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
56.1% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
62.3% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
59.7% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
58.8% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
```



weekday vs KWdelivered

```
# @title weekday vs KWdelivered

from matplotlib import pyplot as plt
import seaborn as sns
figsize = (12, 1.2 * len(df['weekday'].unique()))
plt.figure(figsize=figsize)
sns.violinplot(df, x='kwhTotal', y='weekday', inner='box', palette='Dark2')
sns.despine(top=True, right=True, bottom=True, left=True)
```

```
<ipython-input-21-f22bca7dbb2b>:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `y` variable to `hue`
```



Review of # Stations in LocationID Statistics

```
# prompt: Group by locationId and count the number of unique stationIdn and sum of  kwhTotal  and FacilityType in decending

# Group by locationId
grouped = df.groupby('locationId').agg(
    {'stationId': pd.Series.nunique, 'kwhTotal': 'sum', 'facilityType': lambda x: x.mode()[0] if not x.mode().empty else Non

# Rename columns for clarity
grouped = grouped.rename(columns={
    'stationId': 'unique_station_count',
    'kwhTotal': 'total_kwh',
    'facilityType': 'mode_facility_type'
})

# Sort by total_kwh in descending order
grouped = grouped.sort_values(by='total_kwh', ascending=False)

grouped
```

locationId	unique_station_count	total_kwh	mode_facility_type
493904	2	2805.86	Manufacturing
976902	8	2572.93	R&D
461655	12	2096.62	R&D
868085	6	1948.03	R&D
481066	4	1658.65	Office
928191	8	1083.76	R&D
144857	6	1010.98	Office
503205	4	983.77	R&D
566549	4	753.23	R&D
978130	3	646.39	Office
517854	3	629.85	R&D
814002	3	628.26	Office
948590	2	504.22	Others
202527	6	479.47	R&D
648339	14	319.18	Office
878393	2	313.81	Manufacturing
125372	2	305.56	Office
747048	1	284.92	Manufacturing
751082	2	275.22	Others
399399	4	241.68	Office
620906	4	119.07	R&D
454147	1	23.15	R&D
700367	2	19.05	Office
572514	1	13.36	R&D
310085	1	6.67	Manufacturing

Next steps: Generate code with grouped | View recommended plots | New interactive sheet

Drop Data (If required)

```
# prompt: Drop Rowds with 'Manufacturing'' and 'Others'

# Drop rows where 'facilityType' is 'Manufacturing' or 'Others'
#df = df[~df['facilityType'].isin(['Manufacturing', 'Others'])]
```

Review of LocationId & Stations Counts

```
# prompt: Group by locationId and count the number of unique stationID decending order

# Assuming 'df' is your DataFrame from the previous code

# Group by 'locationId' and count unique 'stationId'
location_station_counts = df.groupby('locationId')['stationId'].nunique().sort_values(ascending=False)

location_station_counts
```

locationId	stationId
648339	14
461655	12
976902	8
928191	8
202527	6
868085	6
144857	6
481066	4
399399	4
620906	4
503205	4
566549	4
517854	3
978130	3
814002	3
493904	2
125372	2
878393	2
948590	2
700367	2
751082	2
454147	1
310085	1
747048	1
572514	1

dtype: int64

```
# prompt: list all the locationId from  location_station_counts

print(location_station_counts.index.tolist())

location_list = location_station_counts.index.tolist()
```

```
[648339, 461655, 976902, 928191, 202527, 868085, 144857, 481066, 399399, 620906, 503205, 566549, 517854, 978130, 814002,
```

```
# prompt: Create a list of all stationId for each locationId in location_list

# Assuming 'df' and 'location_list' are defined as in the previous code.
```

```
station_id_by_location = {}
for location_id in location_list:
  station_ids = df[df['locationId'] == location_id]['stationId'].unique().tolist()
  station_id_by_location[location_id] = station_ids

# Print or use the dictionary as needed
for location, station_ids in station_id_by_location.items():
  print(f"Location ID: {location}, Station IDs: {station_ids}")
```

```
Location ID: 648339, Station IDs: [488364, 540225, 914305, 884707, 922416, 787792, 914907, 481613, 864630, 451479, 28608
Location ID: 461655, Station IDs: [920264, 569889, 878706, 549414, 582873, 371335, 129465, 594591, 612116, 431796, 94376
Location ID: 976902, Station IDs: [944515, 405157, 250527, 280221, 500856, 738900, 801274, 355208]
Location ID: 928191, Station IDs: [944575, 207262, 711566, 728918, 236379, 425476, 256233, 894409]
Location ID: 202527, Station IDs: [507660, 672450, 792421, 911231, 260781, 200695]
Location ID: 868085, Station IDs: [995505, 664306, 932939, 638536, 489543, 569886]
Location ID: 144857, Station IDs: [572249, 951880, 803398, 517464, 568071, 891421]
Location ID: 481066, Station IDs: [474204, 260408, 191826, 562656]
Location ID: 399399, Station IDs: [758780, 280415, 404112, 946482]
Location ID: 620906, Station IDs: [236840, 989457, 623134, 134427]
Location ID: 503205, Station IDs: [653450, 228137, 349635, 693651]
Location ID: 566549, Station IDs: [514669, 902592, 414088, 131897]
Location ID: 517854, Station IDs: [643012, 782629, 729642]
Location ID: 978130, Station IDs: [987396, 366832, 288445]
Location ID: 814002, Station IDs: [386940, 874950, 219054]
Location ID: 493904, Station IDs: [369001, 955429]
Location ID: 125372, Station IDs: [445920, 871619]
Location ID: 878393, Station IDs: [730023, 265601]
Location ID: 948590, Station IDs: [237105, 575971]
Location ID: 700367, Station IDs: [616125, 818217]
Location ID: 751082, Station IDs: [863084, 929628]
Location ID: 454147, Station IDs: [861532]
Location ID: 310085, Station IDs: [300866]
Location ID: 747048, Station IDs: [875343]
Location ID: 572514, Station IDs: [981639]
```

```
# prompt: create a single list of all the station_ids in station_id_by_location.items()

all_station_ids = []
for station_ids in station_id_by_location.values():
  all_station_ids.extend(station_ids)
all_station_ids
```

```
        288445,
        386940,
        874950,
        219054,
        369001,
        955429,
        445920,
        871619,
        730023,
        265601,
        237105,
        575971,
        616125,
        818217,
        863084,
        929628,
        861532,
        300866,
        875343,
        981639]
```

```
# prompt: convert all all_station_ids to integer values

all_station_ids = [int(x) for x in all_station_ids]
all_station_ids
```

```
        200695,
        995505,
        664306,
        932939,
        638536,
        489543,
        569886,
        572249,
        951880,
```

```
875343,
9816391
```

```
# prompt: convert all_station_ids to a dataframe named df_all_station_ids

df_all_station_ids = pd.DataFrame({'station_id': all_station_ids})
df_all_station_ids.head(105)
```

	station_id
0	488364
1	540225
2	914305
3	884707
4	922416
...	...
100	929628
101	861532
102	300866
103	875343
104	981639

105 rows × 1 columns

Next steps: Generate code with df_all_station_ids | View recommended plots | New interactive sheet

```
# prompt: list cells of the dataframe df_all_station_ids that are NaN or 0 or -1

# Find cells that are NaN, 0, or -1
nan_or_zero_or_neg_one_cells = df_all_station_ids[
    (df_all_station_ids['station_id'].isna()) |
    (df_all_station_ids['station_id'] == 0) |
    (df_all_station_ids['station_id'] == -1)
]

nan_or_zero_or_neg_one_cells
```

station_id

```
len(df_all_station_ids)
```

```
105
```

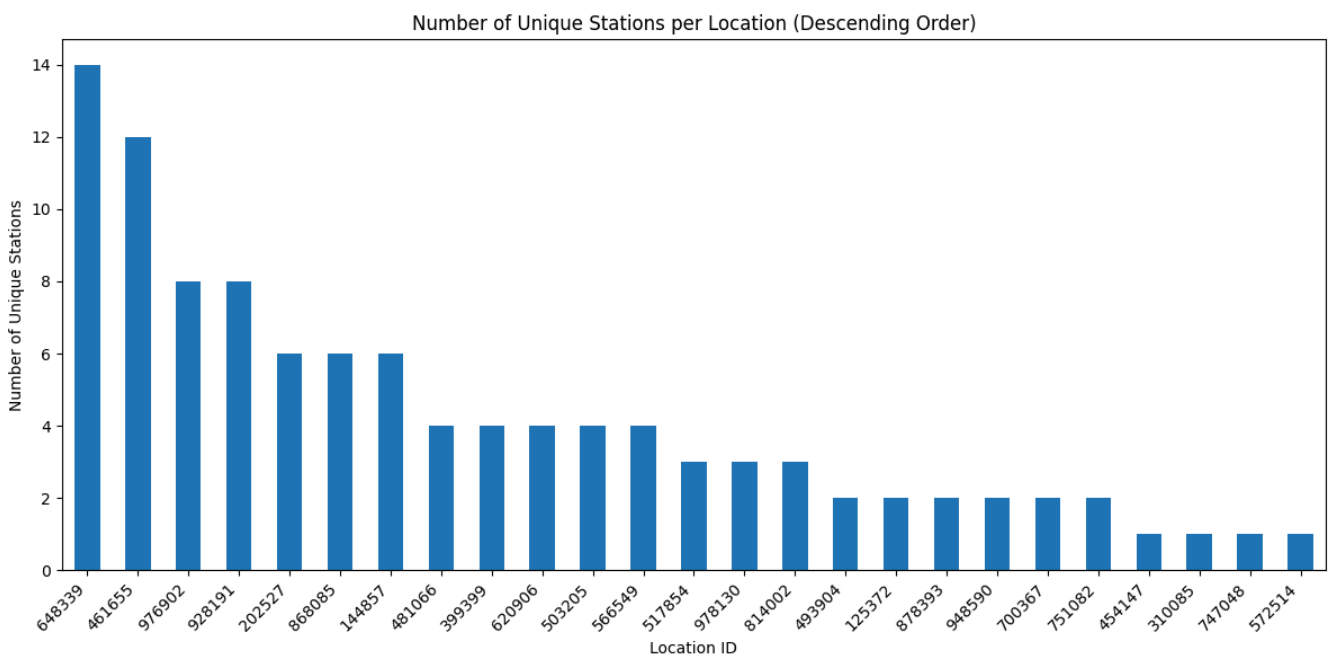
```
# prompt: convert all_stations_ids to integer

all_station_ids = [int(x) for x in all_station_ids]
```

```
# prompt: create a dataframe with the location_station_counts data with columnn stationID listing all the stationID and colu

location_station_counts = df.groupby('locationId')['stationId'].nunique().sort_values(ascending=False)

# Create a DataFrame from the Series
location_station_df = pd.DataFrame({'locationId': location_station_counts.index,
                                    'stationID': location_station_counts.values})

location_station_df
```

	locationId	stationID
0	648339	14
1	461655	12
2	976902	8
3	928191	8
4	202527	6
5	868085	6
6	144857	6
7	481066	4
8	399399	4
9	620906	4
10	503205	4
11	566549	4
12	517854	3
13	978130	3
14	814002	3
15	493904	2
16	125372	2
17	878393	2
18	948590	2
19	700367	2
20	751082	2
21	454147	1
22	310085	1
23	747048	1
24	572514	1

Next steps: Generate code with `location_station_df` View recommended plots New interactive sheet

```
# prompt: Plot Group by locationId and count the number of unique stationIdnin decending order

import matplotlib.pyplot as plt

# Assuming 'location_station_counts' is already calculated as in your provided code

# Plotting the grouped data
plt.figure(figsize=(12, 6))  # Adjust figure size for better visualization
location_station_counts.plot(kind='bar')
plt.title('Number of Unique Stations per Location (Descending Order)')
plt.xlabel('Location ID')
plt.ylabel('Number of Unique Stations')
plt.xticks(rotation=45, ha='right') # Rotate x-axis labels for readability
plt.tight_layout() # Adjust layout to prevent labels from overlapping
plt.show()
```



Review of User Statistics by LocationID

```
# prompt: Group by locationId and count the number of unique userId decending order

# Group by locationId and count the number of unique userId
location_user_counts = df.groupby('locationId')['userId'].nunique().sort_values(ascending=False)

location_user_counts
```

locationId	userId
461655	16
976902	15
868085	14
566549	13
503205	12
202527	11
928191	9
648339	8
493904	7
978130	6
620906	6
144857	5
481066	5
747048	4
751082	3
399399	3
125372	3
572514	3
814002	3
700367	2
454147	2
948590	2
517854	2
310085	1
878393	1

dtype: int64

```
# prompt: create a dataframe called df_userID from all the unique value in df['userId']

df_userID = pd.DataFrame({'userId': df['userId'].unique()})
```

```
# prompt: convert the values of df_userID to integer

# Convert the 'userId' column to integers, handling potential errors
df_userID['userId'] = pd.to_numeric(df_userID['userId'], errors='coerce').astype('Int64')
```

```
len(df_userID)
```

```
85
```

Review of Charge Sessions by LocationID

```
# prompt: Group by locationId and count the number of unique sessionId decending order

# Group by locationId and count the number of unique sessionId
location_session_counts = df.groupby('locationId')['sessionId'].nunique().sort_values(ascending=False)

location_session_counts
```

locationId	sessionId
493904	524
976902	401
461655	393
868085	294
481066	276
928191	235
144857	181
503205	168
566549	158
978130	125
814002	108
202527	103
948590	76
648339	74
747048	48
399399	47
125372	47
517854	47
751082	32
620906	24
878393	20
572514	5
454147	4
700367	4
310085	1

dtype: int64

```
# prompt: Group by locationId  and count of  unique sessionId decending orderand count the number of unique stationid decend

# Group by locationId and aggregate the data
grouped_data = df.groupby('locationId').agg(
    unique_sessions=('sessionId', 'nunique'),
    unique_stations=('stationId', 'nunique'),
    unique_users=('userId', 'nunique'),
    total_kwh=('kwhTotal', 'sum')
)

# Sort the results
sorted_data = grouped_data.sort_values(
    by=['unique_sessions', 'unique_stations', 'unique_users', 'total_kwh'],
    ascending=[False, False, False, False]
)

sorted_data
```

locationId	unique_sessions	unique_stations	unique_users	total_kwh
493904	524	2	7	2805.86
976902	401	8	15	2572.93
461655	393	12	16	2096.62
868085	294	6	14	1948.03
481066	276	4	5	1658.65
928191	235	8	9	1083.76
144857	181	6	5	1010.98
503205	168	4	12	983.77
566549	158	4	13	753.23
978130	125	3	6	646.39
814002	108	3	3	628.26
202527	103	6	11	479.47
948590	76	2	2	504.22
648339	74	14	8	319.18
747048	48	1	4	284.92
399399	47	4	3	241.68
517854	47	3	2	629.85
125372	47	2	3	305.56
751082	32	2	3	275.22
620906	24	4	6	119.07
878393	20	2	1	313.81
572514	5	1	3	13.36
700367	4	2	2	19.05
454147	4	1	2	23.15
310085	1	1	1	6.67

Next steps: Generate code with `sorted_data` View recommended plots New interactive sheet

Review of Energy & Charge time

```
#describe data
df[['kwhTotal','chargeTimeHrs']].describe()
```

	kwhTotal	chargeTimeHrs
count	3395.000000	3395.000000
mean	5.809629	2.841488
std	2.892727	1.507472
min	0.000000	0.012500
25%	4.350000	2.110278
50%	6.230000	2.808889
75%	6.830000	3.544167
max	23.680000	55.238056

Conclusion

As this Data is old, the charge times are considerable larger that the current EV fast charger times.

```
# prompt: Group by locationId and sum of  kwhTotal decending order

# Group by locationId and sum kwhTotal, then sort in descending order
kwh_by_location = df.groupby('locationId')['kwhTotal'].sum().sort_values(ascending=False)
kwh_by_location
```

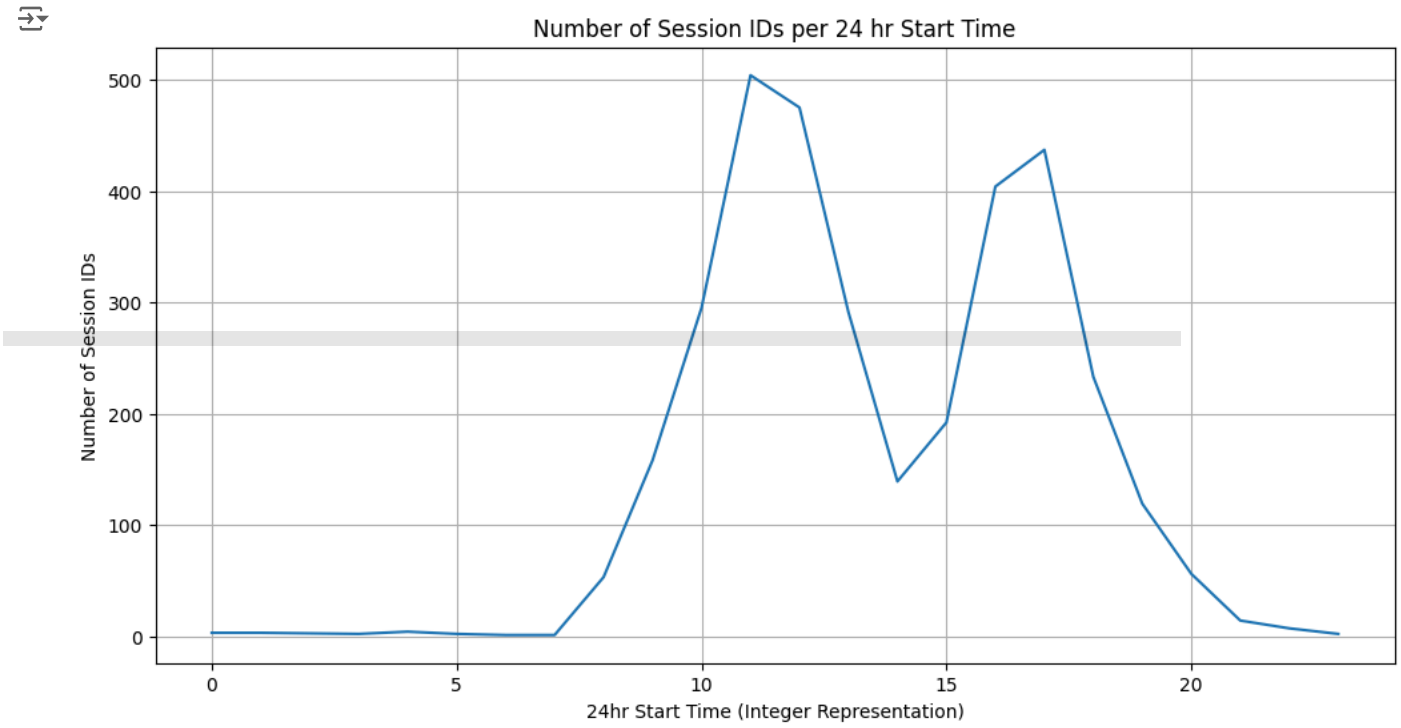
locationId	kwhTotal
493904	2805.86
976902	2572.93
461655	2096.62
868085	1948.03
481066	1658.65
928191	1083.76
144857	1010.98
503205	983.77
566549	753.23
978130	646.39
517854	629.85
814002	628.26
948590	504.22
202527	479.47
648339	319.18
878393	313.81
125372	305.56
747048	284.92
751082	275.22
399399	241.68
620906	119.07
454147	23.15
700367	19.05
572514	13.36
310085	6.67

dtype: float64

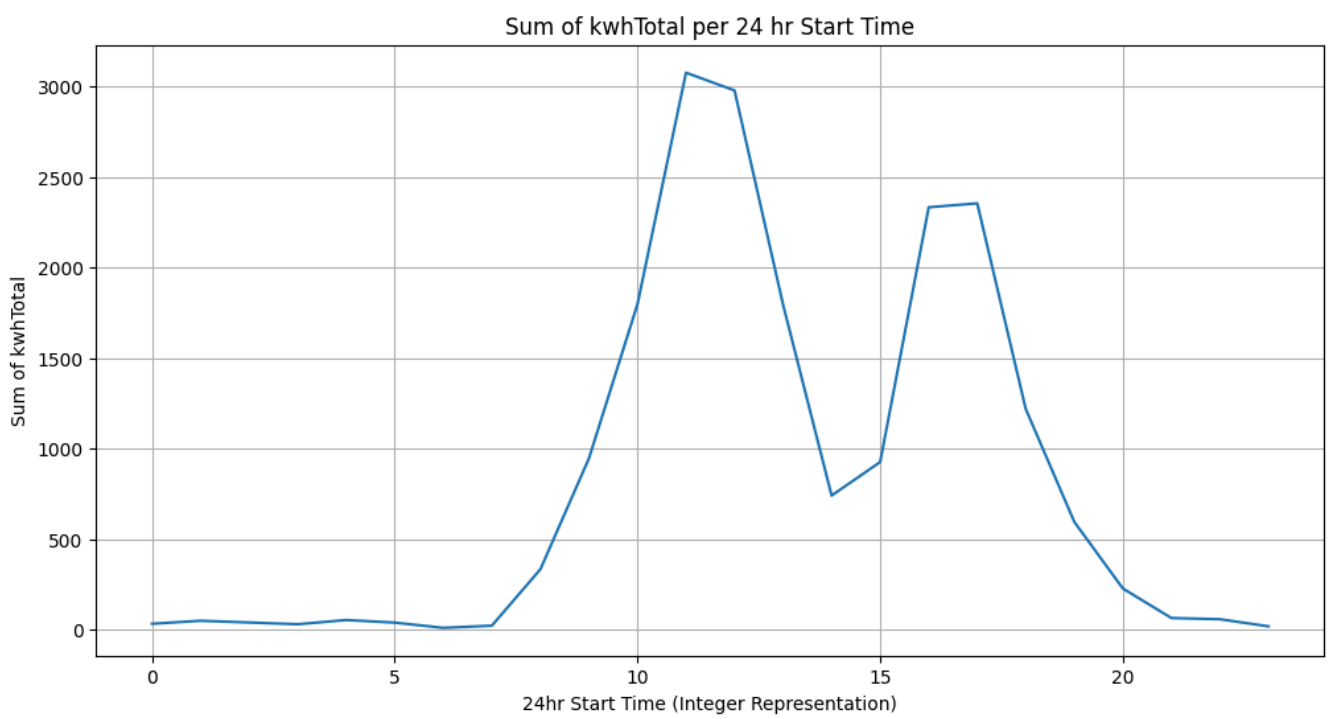
EV Charge 24hr Session Time Dimension Analysis

```
# prompt: Convert startTime to integer and Plot the number of sessionId per startTime

# Convert 'startTime' to datetime objects if it's not already
df['startTime_int'] = df['startTime'].astype(int) # create a new column named startTime_int to hold the converted values fro



# Group by the integer representation of startTime and count the number of sessionIds
session_counts_per_start_time = df.groupby('startTime_int')['sessionId'].count()

# Plotting
plt.figure(figsize=(12, 6))
```

```
plt.plot(session_counts_per_start_time.index, session_counts_per_start_time.values)
plt.xlabel('24hr Start Time (Integer Representation)')
plt.ylabel('Number of Session IDs')
plt.title('Number of Session IDs per 24 hr Start Time')
plt.grid(True)
plt.show()
```

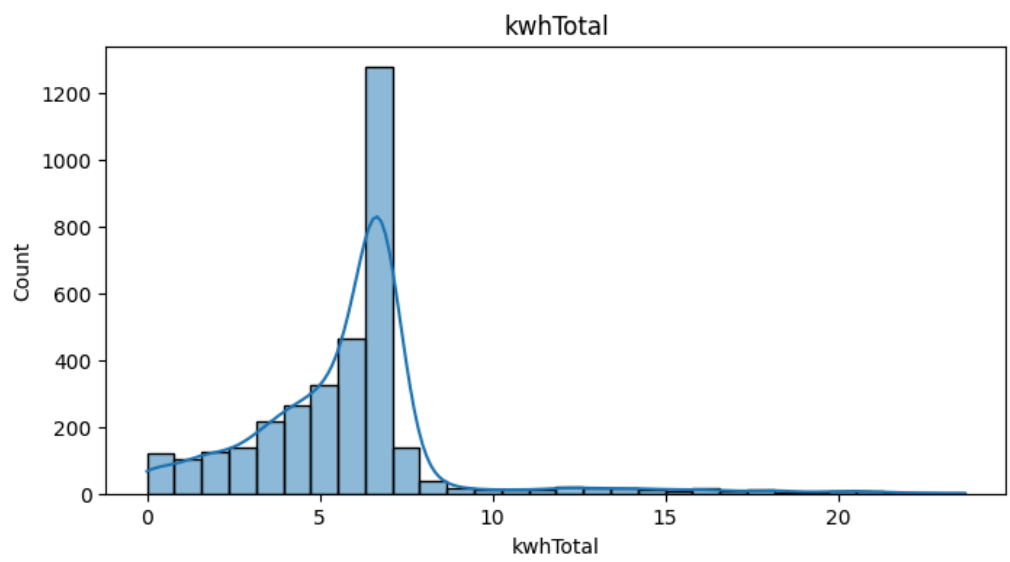



```
# prompt: Convert startTime to integer and Plot the sum of kwhTotal per startTime

# Convert 'startTime' to datetime objects if it's not already
df['startTime_int'] = pd.to_numeric(df['startTime'], errors='coerce').fillna(0).astype(int)

# Group by the integer representation of startTime and sum kwhTotal
kwh_per_start_time = df.groupby('startTime_int')['kwhTotal'].sum()

# Plotting
plt.figure(figsize=(12, 6))
plt.plot(kwh_per_start_time.index, kwh_per_start_time.values)
plt.xlabel('24hr Start Time (Integer Representation)')
plt.ylabel('Sum of kwhTotal')
plt.title('Sum of kwhTotal per 24 hr Start Time')
plt.grid(True)
plt.show()
```

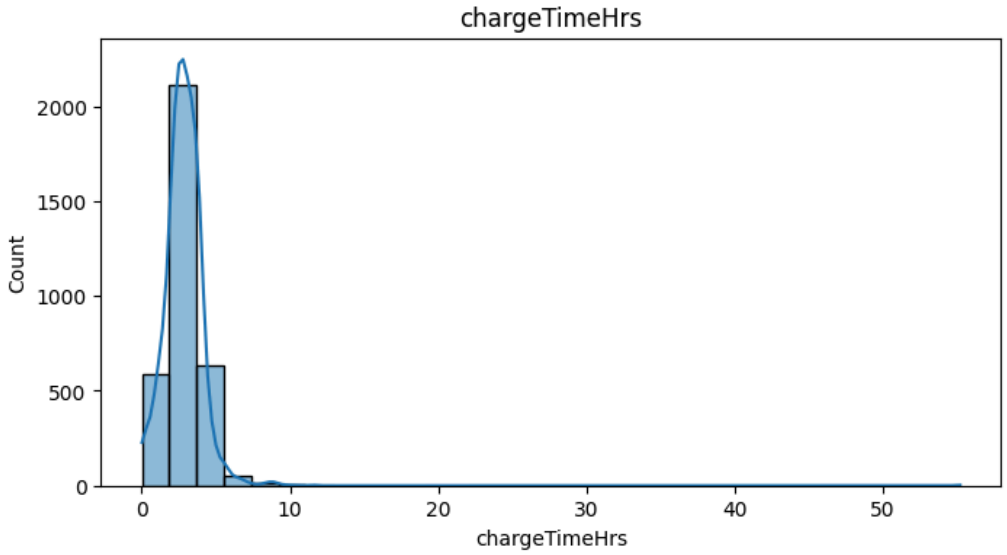


Power Consumption by EV for each Charging Event

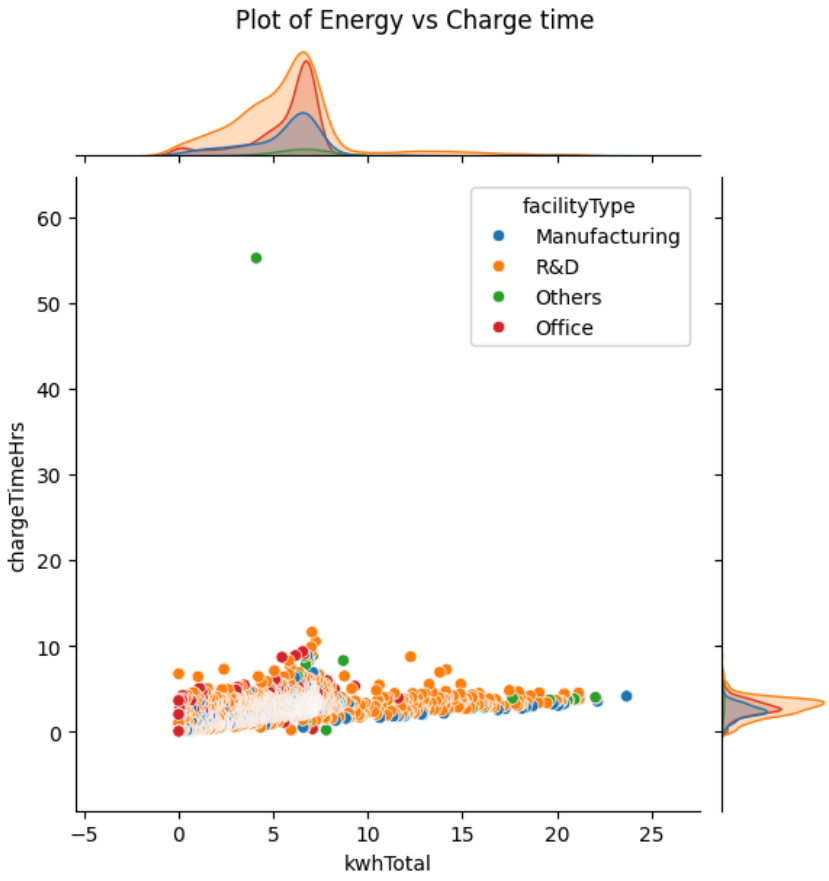
```
import matplotlib.pyplot as plt
import seaborn as sns

#Distribution of Energy Consumed kwhTotal

plt.figure(figsize=(8,4))
sns.histplot(df["kwhTotal"], bins = 30, kde = True)
plt.title("kwhTotal")
plt.show()
```




```
#Distribution of chargeTimeHrs

plt.figure(figsize=(8,4))
sns.histplot(df["chargeTimeHrs"], bins = 30, kde = True)
plt.title("chargeTimeHrs")
plt.show()
```




```
sns.jointplot(data=df, x='kwhTotal', y='chargeTimeHrs', hue = 'facilityType' ) # Removed the extra parenthesis from 'chargeT
plt.suptitle(f'Plot of Energy vs Charge time', y=1.02)
plt.show()
```




```
df.head()
```

	sessionId	kwhTotal	dollars	created	ended	startTime	endTime	chargeTimeHrs	weekday	platform	distance	user
0	4926737	23.68	0.5	0015-10-03 07:18:43	0015-10-03 11:25:07	7	11	4.106667	Sat	android	NaN	789081
1	3738844	22.14	0.0	0015-08-01 05:29:02	0015-08-01 09:00:08	5	9	3.518333	Sat	android	NaN	789081
2	2682332	22.07	0.0	0015-09-02 13:43:27	0015-09-02 17:38:10	13	17	3.911944	Wed	android	NaN	789081
3	9025610	22.03	0.0	0015-05-29 16:55:35	0015-05-29 20:54:06	16	20	3.975278	Fri	android	NaN	789081
4	4473237	21.20	0.0	0015-09-30 16:52:49	0015-09-30 20:42:06	16	20	3.821389	Wed	android	NaN	789081

STATION DATA LOCALISATION TO MELBOURBNE VICTOIRA

Use the following mapping tool to obtain some proximity data (20km radius) in relation to CBD Post code 3000

https://www.freemaptools.com/find-australian-postcodes-inside-radius.htm#google_vignette

Generate a csv with the data.

Load postcode Data and calculate distance to CBD

```
#load dataset csv from https://www.freemaptools.com/find-australian-postcodes-inside-radius.htm#google_vignette
df3= pd.read_csv('Melbourne_PostCodes_Data.csv')
```

```
df3.head(20)
```

	postcode, suburb, distance
0	3000,Melbourne,0.00
1	3003,West Melbourne,0.96
2	3005,World Trade Centre,0.95
3	3006,South Wharf,0.97
4	3008,Docklands,0.82
5	8012,Docklands,0.82
6	3006,Southbank,1.03
7	3050,Royal Melbourne Hospital,1.91
8	3053,Carlton,1.74
9	3053,Carlton South,1.74
10	3205,South Melbourne,1.59
11	3205,South Melbourne Dc,1.59
12	3002,East Melbourne,2.24
13	3010,University Of Melbourne,2.40
14	3051,Hotham Hill,2.20
15	3051,North Melbourne,2.20
16	3065,Fitzroy,2.47
17	3066,Collingwood,2.99
18	3066,Collingwood North,2.99
19	3004,Melbourne,3.21

Next steps: Generate code with df3 View recommended plots New interactive sheet

```
# prompt: split the dataframe df3 on comma separator into a new df4 with columns postcode, suburb and distance

# Access the first column (which likely contains the data) by index (0)
df4 = df3.iloc[:, 0].str.split(',', expand=True)
df4.columns = ['postcode', 'suburb', 'distance']
```

```
# prompt: for each unique postcodes calculate the mean distance and sort by mean distance in ascending order

# Assuming df4 is the DataFrame with 'postcode' and 'distance' columns as created in your provided code.
# Convert the 'distance' column to numeric, handling potential errors
df4['distance'] = pd.to_numeric(df4['distance'], errors='coerce')

# Group by postcode and calculate the mean distance
mean_distances = df4.groupby('postcode')['distance'].mean()

# Sort by mean distance in ascending order
sorted_mean_distances = mean_distances.sort_values(ascending=True)

sorted_mean_distances.head(25)
```

postcode	distance
3000	0.00000
3008	0.82000
8012	0.82000
3005	0.95000
3003	0.96000
3006	1.00000
3205	1.59000
3053	1.74000
3050	1.91000
3051	2.20000
3002	2.24000
3010	2.40000
3065	2.47000
3066	2.99000
3004	3.21000
8008	3.21000
3052	3.23000
3206	3.28000
3054	3.51000
3207	3.78000
3068	3.91000
3067	3.95000
3141	3.98000
3121	4.07625
3031	4.13000

dtype: float64

```
# prompt: convert mean distance to a dataframe called postcode_distance

postcode_distance = pd.DataFrame({'postcode': sorted_mean_distances.index, 'mean_distance': sorted_mean_distances.values})
postcode_distance
```

	postcode	mean_distance
0	3000	0.000
1	3008	0.820
2	8012	0.820
3	3005	0.950
4	3003	0.960
...	...	...
359	3992	97.640
360	3662	97.845
361	3321	99.030
362	3822	99.750
363	3350	99.900

364 rows × 2 columns

Next steps: Generate code with postcode_distance | View recommended plots | New interactive sheet

```
# prompt: print the  unique_location_count and unique_station_count

print(f"The number of unique location IDs is: {unique_location_count}")
print(f"The number of unique station IDs is: {unique_station_count}")
```

```
The number of unique location IDs is: 25
The number of unique station IDs is: 105
```

Load cleaned Station Data

```
#load dataset
df3= pd.read_csv('Melbourne_PostCodes_Data.csv')
```

```
#load dataset
df5= pd.read_csv('Cleaned_Australian_EV_Charging_Stations.csv')
```

```
df5.head(2)
```

	Station_no	Location Name	Latitude	Longitude	Town	Postal Code	City	Address	Plugs_Type2	Plugs_Three_Phase	Plugs_CHA
0	0	Chargefox Charging Station	-35.726720	145.659354	Cobram	2714	N.A	154 Campbell Rd	1.0	NaN	
1	82	Evie Charging Station	-35.940484	144.725516	Echuca	2731	N.A	438 High St	0.0	NaN	

```
# prompt: from df5 remove rows with Postal Code = N.A and remove rows with Postal Code < 3000 and remove rows with Postal Co

# Assuming df5 is already loaded as in your provided code.
# Filter out rows where 'Postal Code' is 'N.A'
df5 = df5[df5['Postal Code'] != 'N.A']

# Convert 'Postal Code' to numeric, handling errors
df5['Postal Code'] = pd.to_numeric(df5['Postal Code'], errors='coerce')

# Filter out rows where 'Postal Code' is less than 3000 or greater than 3999
df5 = df5[(df5['Postal Code'] >= 3000) & (df5['Postal Code'] <= 3999)]
```

```
df5.tail(5)
```

	Station_no	Location Name	Latitude	Longitude	Town	Postal Code	City	Address	Plugs_Type2	Plugs_Three_Phase	P
336	210	Sorrento Community Centre	-38.338118	144.737278	N.A	3943	Melbourne	860 Melbourne Rd, Sorrento VIC 3943, Australia	NaN	NaN	
337	390	Sorrento Hotel	-38.338316	144.743447	N.A	3943	Melbourne	1 Esplanade, Sorrento VIC 3943, Australia	0.0	0.0	
338	300	Cranbourne West Community Hub (Coming Soon)	-38.106929	145.250522	N.A	3977	Melbourne	75 Evans Rd, Cranbourne West VIC 3977, Australia	2.0	0.0	
339	308	Cranbourne Home (Out Of Service)	-38.087041	145.279599	N.A	3977	Melbourne	2 Lesdon Ave, Cranbourne VIC 3977, Australia	2.0	0.0	
340	120	Club Delaray	-38.110037	145.336394	N.A	3978	Melbourne	20 Alphey Road, Clyde North VIC 3978, Australia	1.0	0.0	

```
# prompt: for every row of df5, using the integer value of df5['Postal Code'] , find the corresponding integer value from da

# Create the 'cbd_distance' column in df5 and initialize it to 0
df5['cbd_distance'] = 0

# Iterate through rows of df5
for index, row in df5.iterrows():
  try:
    # Find the corresponding row in df4 based on 'Postal Code'
    postal_code = int(row['Postal Code'])
    matching_row = df4[df4['postcode'] == str(postal_code)]

    # If a match is found, add the 'distance' to 'cbd_distance' in df5
    if not matching_row.empty:
      df5.loc[index, 'cbd_distance'] = matching_row['distance'].iloc[0]
  except (ValueError, KeyError):
    # Handle cases where 'Postal Code' is not an integer or not found in df4
    print(f"Error processing row {index}: Invalid Postal Code or not found in df4")
    # You might want to set a default value or skip the row based on your needs
```

```
<ipython-input-60-f73cefbf5930>:15: FutureWarning:

Setting an item of incompatible dtype is deprecated and will raise an error in a future version of pandas. Value '2.24'
```

```
df5.head(5)
```

	Station_no	Location Name	Latitude	Longitude	Town	Postal Code	City	Address	Plugs_Type2	Plugs_Three_Phase	Plugs_
2	2	Lonsdale St	-37.813437	144.955934	N.A	3000	Melbourne	535 Little Lonsdale St, Melbourne VIC 3004, Au...	0.0	0.0	
3	22	Bourke St	-37.814779	144.956447	N.A	3000	Melbourne	542 Little Bourke St, Melbourne VIC 3000, Aust...	0.0	0.0	
4	34	Secure Parking (450 Flinders Lane)	-37.818758	144.958866	N.A	3000	Melbourne	450 Flinders Ln, Melbourne VIC 3000, Australia	10.0	0.0	
5	42	Hyatt Centric Melbourne	-37.820407	144.955802	N.A	3000	Melbourne	25 Downie St, Melbourne VIC 3000, Australia	2.0	0.0	
6	55	Wilson Parking	-37.811294	144.967823	N.A	3000	Melbourne	222 Russell St, Melbourne VIC 3000, Australia	2.0	0.0	

```
# prompt: for every row of df5, add a column to df5 called ChargingPoints that is the sum of Plugs_Type2 + Plugs_Three_Phase

# Assuming df5 is already loaded and processed as in the provided code.

# Fill NaN values in relevant columns with 0
for col in ['Plugs_Type2', 'Plugs_Three_Phase', 'Plugs_CHAdeMO', 'Plugs_CCS/SAE', 'Plugs_Tesla']:
    df5[col] = df5[col].fillna(0)

# Calculate 'ChargingPoints' for each row
df5['ChargingPoints'] = df5['Plugs_Type2'] + df5['Plugs_Three_Phase'] + df5['Plugs_CHAdeMO'] + df5['Plugs_CCS/SAE'] + df5['P
```

```
# prompt: for cbd_distance < 10 count the unique Station_no

# Assuming df5 is already loaded and processed as in the provided code.

# Filter for cbd_distance < 10
filtered_df = df5[df5['cbd_distance'] < 10]

# Count unique Station_no (assuming 'Station_no' is a column name)
unique_station_count = filtered_df['Station_no'].nunique()

print(f"Number of unique stations with cbd_distance < 10: {unique_station_count}")
```

```
Number of unique stations with cbd_distance < 10: 141
```

```
# prompt: list count the unique Station_no descending for Postal Code and cbd_distance

# Assuming df5 is already loaded and processed as in the provided code.

# Group by 'Postal Code' and 'cbd_distance', then count unique 'Station_no'
station_counts = df5.groupby(['Postal Code', 'cbd_distance'])['Station_no'].nunique().sort_values(ascending=False)

station_counts
```

Postal Code	cbd_distance	Station_no
3000	**0.00**	40
3207	**3.78**	11
3205	**1.59**	7
3121	**4.01**	7
3931	**46.59**	6
...	**...**	...
3178	**27.69**	1
3179	**24.97**	1
3162	**10.16**	1
3165	**14.74**	1
3167	**16.45**	1

147 rows × 1 columns

dtype: int64

```
# prompt: sort df5 in ascending order base on column cbd_distance

# Sort df5 in ascending order based on the 'cbd_distance' column
df5_sorted = df5.sort_values(by='cbd_distance', ascending=True)
df5_sorted.head()
```

	Station_no	Location Name	Latitude	Longitude	Town	Postal Code	City	Address	Plugs_Type2	Plugs_Three_Phase
282	370	Electric Vehicle Charging Station	-36.385785	145.396965	Shepparton	3630	Shepparton	530 Wyndham St	1.0	0.0
24	227	Downie St	-37.820203	144.955565	N.A	3000	Melbourne	25 Downie St, Melbourne VIC 3000, Australia	2.0	0.0
23	219	Secure Parking (Southern Cross)	-37.812452	144.969535	N.A	3000	Melbourne	129 Bourke St, Melbourne VIC 3000, Australia	0.0	0.0
22	214	Secure Parking - 460 Lonsdale Street Car Park	-37.813049	144.958347	N.A	3000	Melbourne	460 Lonsdale St, Melbourne VIC 3000, Australia	0.0	0.0
21	205	Wilson Parking	-37.814781	144.956451	N.A	3000	Melbourne	542 Little Bourke St, Melbourne VIC 3000, Aust...	0.0	0.0

```
# prompt: remove from df5 rows with ChargingPoints=0

# Assuming df5 is already loaded and processed as in the provided code.

# Remove rows where 'ChargingPoints' is equal to 0
df5 = df5[df5['ChargingPoints'] != 0]
df5.tail()
```

	Station_no	Location Name	Latitude	Longitude	Town	Postal Code	City	Address	Plugs_Type2	Plugs_Three_Phase	P
334	3	Sorrento Community Centre	-38.338560	144.737043	N.A	3943	Melbourne	860 Melbourne Rd, Sorrento VIC 3943, Australia	0.0	0.0	
336	210	Sorrento Community Centre	-38.338118	144.737278	N.A	3943	Melbourne	860 Melbourne Rd, Sorrento VIC 3943, Australia	0.0	0.0	
338	300	Cranbourne West Community Hub (Coming Soon)	-38.106929	145.250522	N.A	3977	Melbourne	75 Evans Rd, Cranbourne West VIC 3977, Australia	2.0	0.0	
339	308	Cranbourne Home (Out Of Service)	-38.087041	145.279599	N.A	3977	Melbourne	2 Lesdon Ave, Cranbourne VIC 3977, Australia	2.0	0.0	
340	120	Club Delaray	-38.110037	145.336394	N.A	3978	Melbourne	20 Alphey Road, Clyde North VIC 3978, Australia	1.0	0.0	

```
# prompt: remove from df5 rows with ChargingPoints= NaN

# Assuming df5 is already loaded and processed as in the provided code.

# Remove rows where 'ChargingPoints' is NaN
df5 = df5.dropna(subset=['ChargingPoints'])
df5.tail()
```

	Station_no	Location Name	Latitude	Longitude	Town	Postal Code	City	Address	Plugs_Type2	Plugs_Three_Phase	P
334	3	Sorrento Community Centre	-38.338560	144.737043	N.A	3943	Melbourne	860 Melbourne Rd, Sorrento VIC 3943, Australia	0.0	0.0	
336	210	Sorrento Community Centre	-38.338118	144.737278	N.A	3943	Melbourne	860 Melbourne Rd, Sorrento VIC 3943, Australia	0.0	0.0	
338	300	Cranbourne West Community Hub (Coming Soon)	-38.106929	145.250522	N.A	3977	Melbourne	75 Evans Rd, Cranbourne West VIC 3977, Australia	2.0	0.0	
339	308	Cranbourne Home (Out Of Service)	-38.087041	145.279599	N.A	3977	Melbourne	2 Lesdon Ave, Cranbourne VIC 3977, Australia	2.0	0.0	
340	120	Club Delaray	-38.110037	145.336394	N.A	3978	Melbourne	20 Alphey Road, Clyde North VIC 3978, Australia	1.0	0.0	

Merge the Station Data & Postcode Data

```
# count numner of unique stationId in df ( the original data)
unique_station_count = len(df_all_station_ids)
print(f"The number of unique station IDs is: {unique_station_count}")
```

```
The number of unique station IDs is: 105
```

```
# prompt: truncate df5 to the length of unique_station_count

# Assuming unique_station_count is already defined and df5 is your DataFrame

df5 = df5.head(unique_station_count)
df5.tail(5)
```

	Station_no	Location Name	Latitude	Longitude	Town	Postal Code	City	Address	Plugs_Type2	Plugs_Three_Ph
131	217	Uni Hill Town Centre	-37.682471	145.071000	N.A	3082	Melbourne	Shop 17/5 Janefield Dr, Bundoora VIC 3083, Aus...	0.0	
132	270	Uni Hill Town Centre	-37.683069	145.071032	N.A	3082	Melbourne	Shop 17/5 Janefield Dr, Bundoora VIC 3083, Aus...	0.0	
134	66	Woolworths Heidelberg	-37.758997	145.068632	N.A	3084	Melbourne	451 Lower Heidelberg Rd, Heidelberg VIC 3084, ...	16.0	
135	169	Baptcare Strathalan	-37.728819	145.077998	N.A	3084	Melbourne	2-34 Erskine Rd, Macleod VIC 3085, Australia	2.0	
136	61	Greensborough Plaza	-37.702301	145.102344	N.A	3088	Melbourne	25 Main St, Greensborough VIC 3088, Australia	4.0	

```
#Chech the number of stationsID
len(df5), unique_station_count
```

```
(105, 105)
```

```
# prompt: add df_all_station_ids as a column in df5 name stationID

df5['stationID'] = df_all_station_ids['station_id'].values[:len(df5)]
df5.head()
```

	Station_no	Location Name	Latitude	Longitude	Town	Postal Code	City	Address	Plugs_Type2	Plugs_Three_Phase	Plugs_
2	2	Lonsdale St	-37.813437	144.955934	N.A	3000	Melbourne	535 Little Lonsdale St, Melbourne VIC 3004, Au...	0.0	0.0	
4	34	Secure Parking (450 Flinders Lane)	-37.818758	144.958866	N.A	3000	Melbourne	450 Flinders Ln, Melbourne VIC 3000, Australia	10.0	0.0	
5	42	Hyatt Centric Melbourne	-37.820407	144.955802	N.A	3000	Melbourne	25 Downie St, Melbourne VIC 3000, Australia	2.0	0.0	
6	55	Wilson Parking	-37.811294	144.967823	N.A	3000	Melbourne	222 Russell St, Melbourne VIC 3000, Australia	2.0	0.0	
7	71	Alto Hotel	-37.816254	144.954636	N.A	3000	Melbourne	636 Bourke St, Melbourne VIC 3000, Australia	0.0	1.0	

```
# prompt: list the unique stations in df5[stationID]
```

```
unique_stations = df5['stationID'].unique()
unique_stations
```

```
array([488364, 540225, 914305, 884707, 922416, 787792, 914907, 481613,
       864630, 451479, 286084, 587514, 988981, 926676, 920264, 569889,
       878706, 549414, 582873, 371335, 129465, 594591, 612116, 431796,
       943765, 632920, 944515, 405157, 250527, 280221, 500856, 738900,
       801274, 355208, 944575, 207262, 711566, 728918, 236379, 425476,
       256233, 894409, 507660, 672450, 792421, 911231, 260781, 200695,
       995505, 664306, 932939, 638536, 489543, 569886, 572249, 951880,
       803398, 517464, 568071, 891421, 474204, 260408, 191826, 562656,
       758780, 280415, 404112, 946482, 236840, 989457, 623134, 134427,
       653450, 228137, 349635, 693651, 514669, 902592, 414088, 131897,
       643012, 782629, 729642, 987396, 366832, 288445, 386940, 874950,
       219054, 369001, 955429, 445920, 871619, 730023, 265601, 237105,
       575971, 616125, 818217, 863084, 929628, 861532, 300866, 875343,
       981639])
```

```
# prompt: delete column df5[locationID]

# Assuming df5 is already defined as in your provided code.

# Delete the 'locationID' column if it exists
if 'locationID' in df5.columns:
    del df5['locationID']
```

```
# prompt: create a column df5[locationID}. for each row of df5[stationID] find the corresponding locationID from station_id_

# Assuming df5 and station_id_by_location are defined as in your provided code.

# Create an empty list to store location IDs
location_ids = []

# Iterate through station IDs in df5
for station_id in df5['stationID']:
  found_location = False
  # Search for the station ID in the station_id_by_location dictionary
  for location, station_list in station_id_by_location.items():
    if station_id in station_list:
      location_ids.append(location)
      found_location = True
      break  # Exit the inner loop once a match is found
  if not found_location:
    location_ids.append(0)  # Append 0 if no match is found

# Add the location IDs as a new column in df5
df5['locationID'] = location_ids
```

```
# prompt: list the  uniue value in df5[locatioID]

unique_location_ids = df5['locationID'].unique()
unique_location_ids, len (unique_location_ids)
```

```
(array([648339, 461655, 976902, 928191, 202527, 868085, 144857, 481066,
        399399, 620906, 503205, 566549, 517854, 978130, 814002, 493904,
        125372, 878393, 948590, 700367, 751082, 454147, 310085, 747048,
        572514]),
 25)
```

```
df5.head(2)
```

	Station_no	Location Name	Latitude	Longitude	Town	Postal Code	City	Address	Plugs_Type2	Plugs_Three_Phase	Plugs_
2	2	Lonsdale St	-37.813437	144.955934	N.A	3000	Melbourne	535 Little Lonsdale St, Melbourne VIC 3004, Au...	0.0	0.0	
4	34	Secure Parking (450 Flinders Lane)	-37.818758	144.958866	N.A	3000	Melbourne	450 Flinders Ln, Melbourne VIC 3000, Australia	10.0	0.0	

```
# prompt: change the name of df5[Power 1] to df5[PowerOutput]

df5 = df5.rename(columns={'Power 1': 'PowerOutput'})
```

```
# prompt: create a new column called df5[Amenities] and for each row randomly select a number of amenities from the followin

import random

amenities_list = ['Coffee', 'Dining', 'Restrooms', 'Shopping', 'Grocery', 'WiFi', 'Valet Parking', 'Car Cleaning']

# Function to randomly select amenities
def random_amenities(n):
  num_amenities = random.randint(1, len(amenities_list))  # Select 1 to all amenities
  selected_amenities = random.sample(amenities_list, num_amenities)
  return ', '.join(selected_amenities)

# Apply the function to create the new column
df5['Amenities'] = [random_amenities(i) for i in range(len(df5))]
```

```
df5.head(2)
```

	Station_no	Location Name	Latitude	Longitude	Town	Postal Code	City	Address	Plugs_Type2	Plugs_Three_Phase	Plugs_
2	2	Lonsdale St	-37.813437	144.955934	N.A	3000	Melbourne	535 Little Lonsdale St, Melbourne VIC 3004, Au...	0.0	0.0	
4	34	Secure Parking (450 Flinders Lane)	-37.818758	144.958866	N.A	3000	Melbourne	450 Flinders Ln, Melbourne VIC 3000, Australia	10.0	0.0	

CURRENT EV VECHICLES

Data on electric vehicles can be obtained from:

https://www.aaa.asn.au/research-data/electric-vehicle/

```
#load dataset
df2= pd.read_csv('Vehicle-Specifications-Sep24.csv')
```

Exporatory Analysis

```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 107 entries, 0 to 106
Data columns (total 9 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   VEHICLE TYPE               107 non-null    object
 1   FUEL TYPE                  107 non-null    object
 2   MODEL                      107 non-null    object
 3   VARIANT DETAILS            107 non-null    object
 4   LISTED PRICE ($AUD)        97 non-null     object
 5   FAST CHARGE TIME (minutes) 80 non-null     object
 6   ANCAP RATING               107 non-null    object
 7   RANGE (km)                 92 non-null     float64
 8   ENERGY CONSUMPTION         92 non-null     float64
dtypes: float64(2), object(7)
memory usage: 7.6+ KB
```

```
df2.head(5)
```

	VEHICLE TYPE	FUEL TYPE	MODEL	VARIANT DETAILS	LISTED PRICE ($AUD)	FAST CHARGE TIME (minutes)	ANCAP RATING	RANGE (km)	ENERGY CONSUMPTION
0	Large SUV	BEV	Audi e-tron	Audi e-tron 55 Auto quattro	NaN	85 mins (5%-80% charge, 50kW charger)	5 star, 2019	459.0	23.2
1	Large Car	BEV	Audi e-tron GT	2024 Audi e-tron GT Auto e-quattro MY24	181784	23 mins (5%-80% charge, 270kW charger)	Unrated	540.0	19.2
2	Large Car	BEV	BMW 5 Series BEV	2024 BMW i5 eDrive40 M Sport G60 Auto	155900	30 mins (10%-80% charge, 205kW charger)	5 star, 2023	550.0	16.5
3	Medium Car	BEV	BMW i4	2024 BMW i4 eDrive35 M Sport G26 Auto	85900	31 mins (10%-80% charge, 205kW charger)	4 star, 2022	520.0	22.2
4	Large Car	BEV	BMW i7	2024 BMW i7 xDrive60 M Sport G70 Auto AWD	306900	34 mins (10%-80% charge, 195kW charger)	Unrated	625.0	22.2

Next steps: Generate code with df2 | View recommended plots | New interactive sheet

Manufacturers Analysis

```
# prompt: Create a new Manufacturer column in df2 by stripping the characters after the first space form the model column

# Assuming df2 is already loaded as shown in the provided code.

# Correct the split method call to not use str accessor directly.
df2['Manufacturer'] = df2['MODEL'].apply(lambda x: x.split(' ', 1)[0])
```

```
df2.tail(5)
```

	VEHICLE TYPE	FUEL TYPE	MODEL	VARIANT DETAILS	LISTED PRICE ($AUD)	FAST CHARGE TIME (minutes)	ANCAP RATING	RANGE (km)	ENERGY CONSUMPTION	Manufacturer
102	Large Car	PHEV	Porsche Panamera PHEV	2024 Porsche Panamera 4 E-Hybrid 972 Auto AWD ...	268100	NaN	Unrated	56.0	19.5	Porsche
103	Large SUV	PHEV	Volkswagen Touareg PHEV	2024 Volkswagen Touareg R RC Auto 4MOTION MY24	129990	NaN	Unrated	51.0	21.1	Volkswagen
104	Medium Car	PHEV	Volvo S60 PHEV	2024 Volvo S60 Recharge T8 Black Edition Auto ...	NaN	300 mins (full charge, 3.6kW charger)	Unrated	90.0	18.6	Volvo
105	Medium SUV	PHEV	Volvo XC60 PHEV	2024 Volvo XC60 Recharge Plus T8 Plug-In Hybri...	92390	300 mins (recharge to 100%, 3.6kW charger)	Unrated	81.0	26.2	Volvo
	Large		Volvo XC90	2024 Volvo XC90		300 mins				

Drop NaN values

```
# prompt: remove rows where RANGE (km) or ENERGY CONSUMPTION or FAST CHARGE TIME (minutes) are Nan

# Drop rows with NaN values in specified columns from df2 (not df)
df2 = df2.dropna(subset=['RANGE (km)', 'ENERGY CONSUMPTION', 'FAST CHARGE TIME (minutes)'])
```

```
#display columns
df2.columns
```

```
Index(['VEHICLE TYPE', 'FUEL TYPE', 'MODEL', 'VARIANT DETAILS',
       'LISTED PRICE ($AUD)', 'FAST CHARGE TIME (minutes)', 'ANCAP RATING',
       'RANGE (km)', 'ENERGY CONSUMPTION', 'Manufacturer'],
      dtype='object')
```

Calculate an estimate for the Battery capacity from the given parameters

Use the vehicle RANGE (KM) and the ENERGY CONSUMPTION (kWh/100km)

```
# prompt: Create a new Battery_capacity column by multiplying RANGE (km) by ENERGY CONSUMPTION by 1/100

# Assuming df2 is already loaded and processed as in the previous code.

df2['Battery_capacity_kW'] = df2['RANGE (km)'] * df2['ENERGY CONSUMPTION'] * (1/100)
```

```
<ipython-input-87-6a33ead56c10>:5: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
```

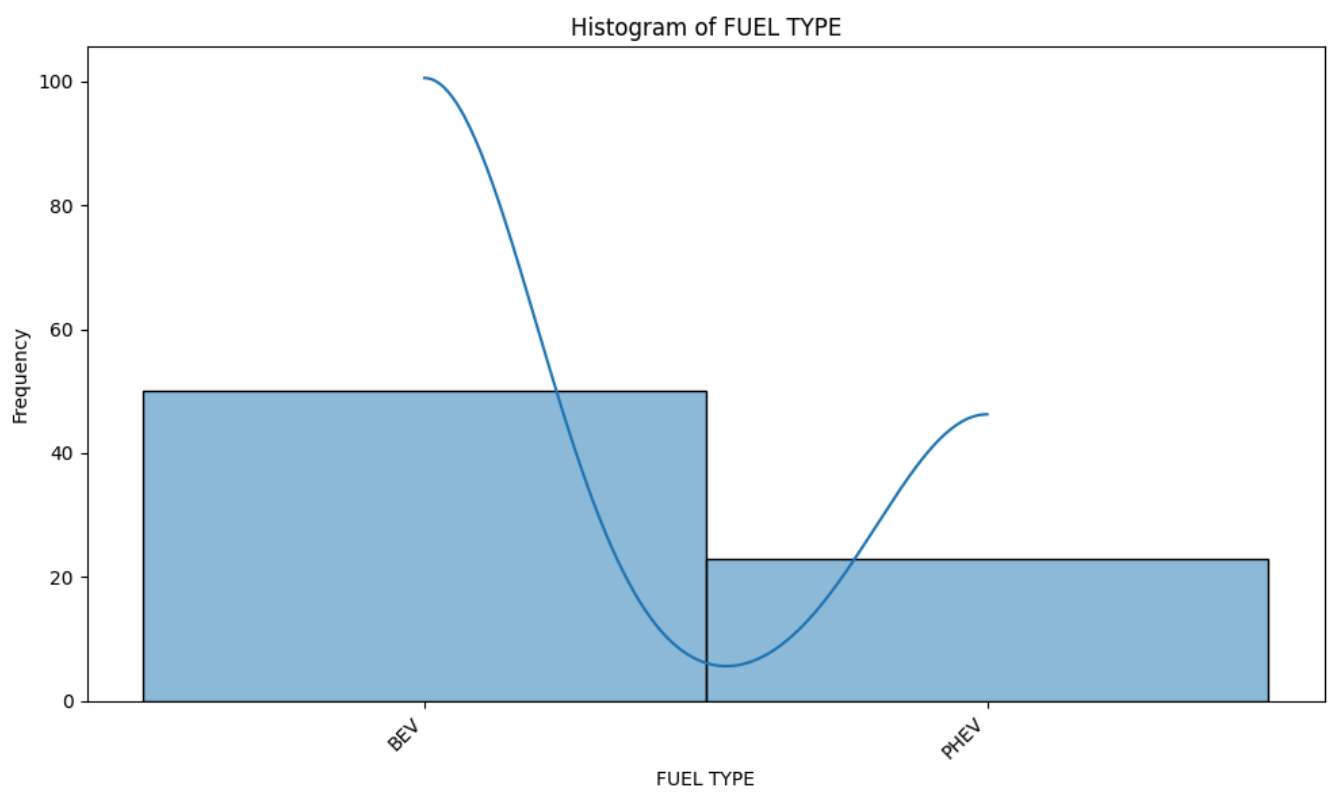
```
df2.head(5)
```

	VEHICLE TYPE	FUEL TYPE	MODEL	VARIANT DETAILS	LISTED PRICE ($AUD)	FAST CHARGE TIME (minutes)	ANCAP RATING	RANGE (km)	ENERGY CONSUMPTION	Manufacturer	Battery_capacity_kW
0	Large SUV	BEV	Audi e-tron	Audi e-tron 55 Auto quattro	NaN	85 mins (5%-80% charge, 50kW charger)	5 star, 2019	459.0	23.2	Audi	106.488
1	Large Car	BEV	Audi e-tron GT	2024 Audi e-tron GT Auto e-quattro MY24	181784	23 mins (5%-80% charge, 270kW charger)	Unrated	540.0	19.2	Audi	103.680
2	Large Car	BEV	BMW 5 Series BEV	2024 BMW i5 eDrive40 M Sport G60 Auto	155900	30 mins (10%-80% charge, 205kW charger)	5 star, 2023	550.0	16.5	BMW	90.750
3	Medium Car	BEV	BMW i4	2024 BMW i4 eDrive35 M Sport G26 Auto	85900	31 mins (10%-80% charge, 205kW charger)	4 star, 2022	520.0	22.2	BMW	115.440
				2024 BMW i7		34 mins					

Next steps: Generate code with df2 | View recommended plots | New interactive sheet

Remove Fuel types other that Battery Electric Vehicles

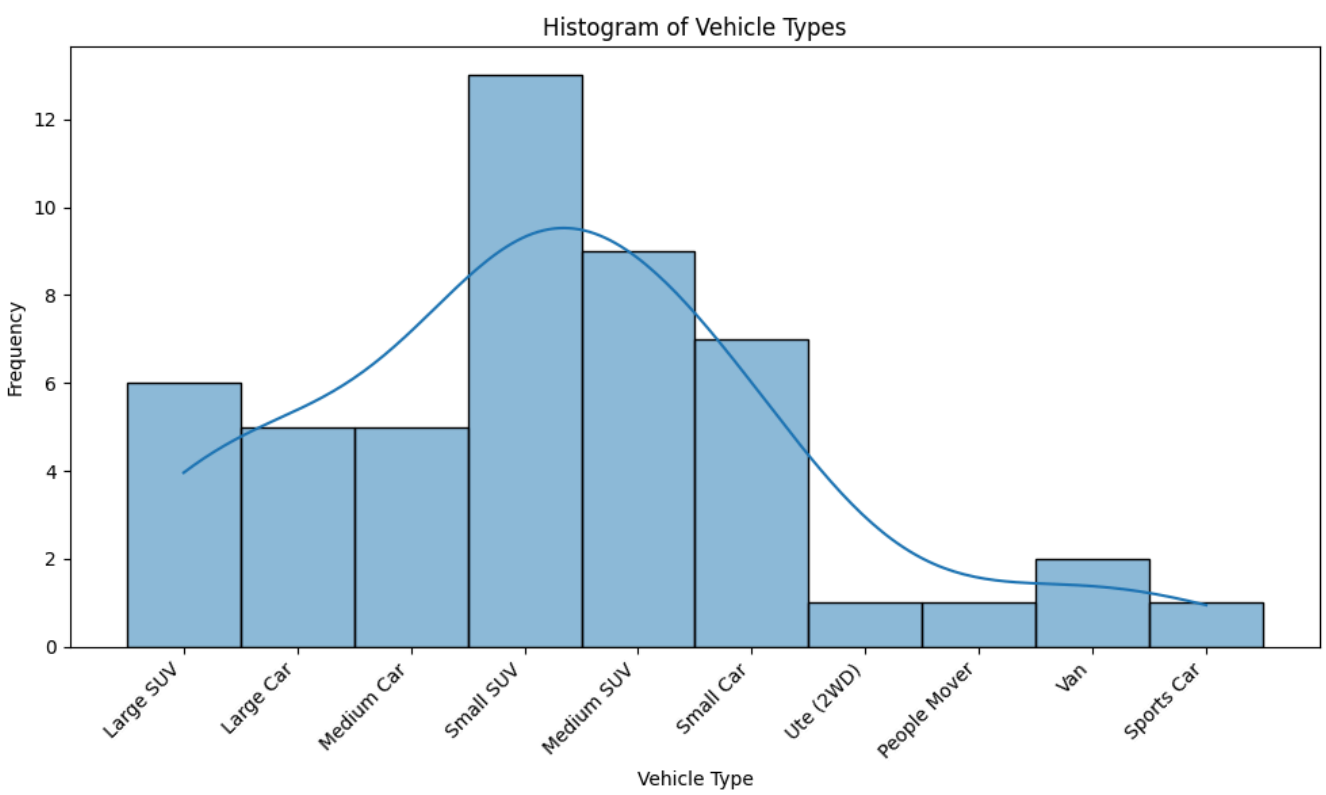
```
# prompt: Plot a histogram of FUEL TYPE

import matplotlib.pyplot as plt
import seaborn as sns

# Assuming df2 is already loaded and processed as in the previous code.

plt.figure(figsize=(10, 6))
sns.histplot(df2['FUEL TYPE'], bins=30, kde=True)
plt.title("Histogram of FUEL TYPE")
plt.xlabel("FUEL TYPE")
plt.ylabel("Frequency")
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for readability
plt.tight_layout()  # Adjust layout to prevent labels from overlapping
plt.show()
```




```
# prompt: drop FUEL TYPE = PHEV from column FUEL TYPE

# Assuming df2 is already loaded and processed as in the previous code.

df2 = df2[df2['FUEL TYPE'] != 'PHEV']
```

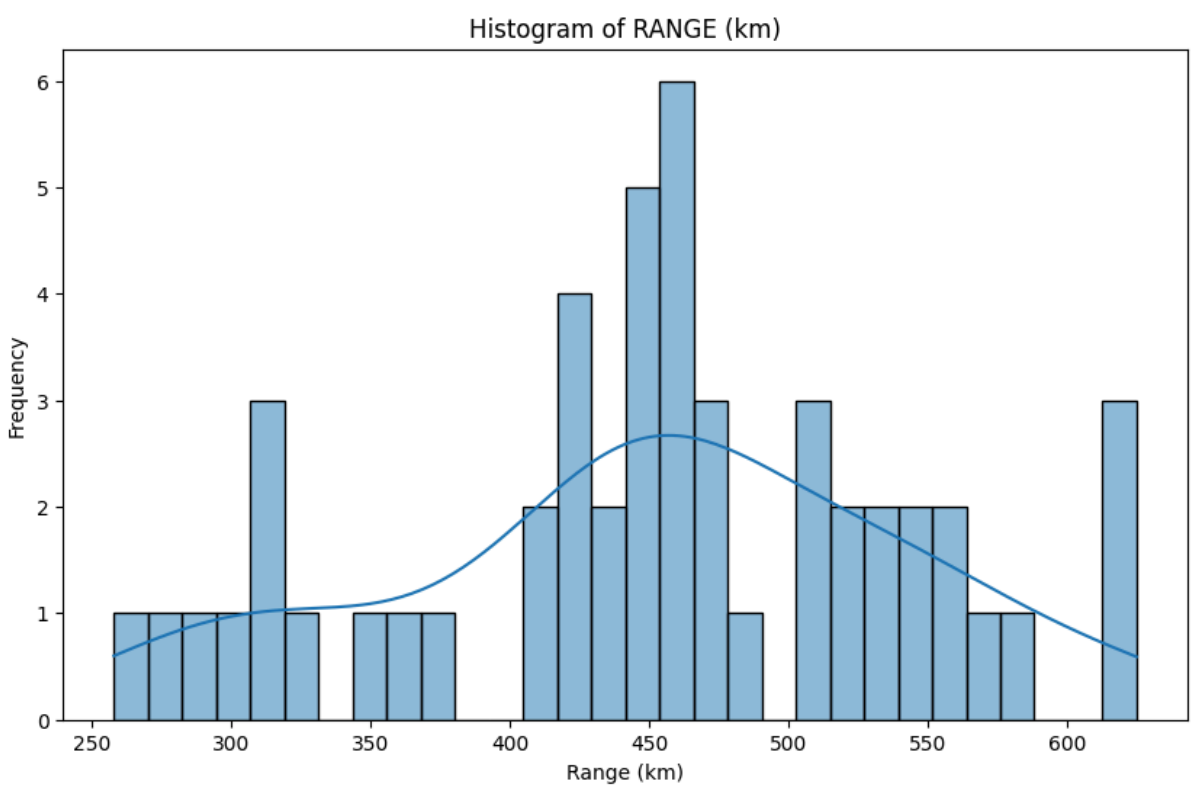
Review distribution of vehicle types or Classes

```
# prompt: Plot a histogram of VEHICLE TYPE

import matplotlib.pyplot as plt
import seaborn as sns

# Assuming df2 is already loaded and processed as in the previous code.

plt.figure(figsize=(10, 6))
sns.histplot(df2['VEHICLE TYPE'], bins=30, kde=True)
plt.title("Histogram of Vehicle Types")
plt.xlabel("Vehicle Type")
plt.ylabel("Frequency")
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for readability
plt.tight_layout()  # Adjust layout to prevent labels from overlapping
plt.show()
```



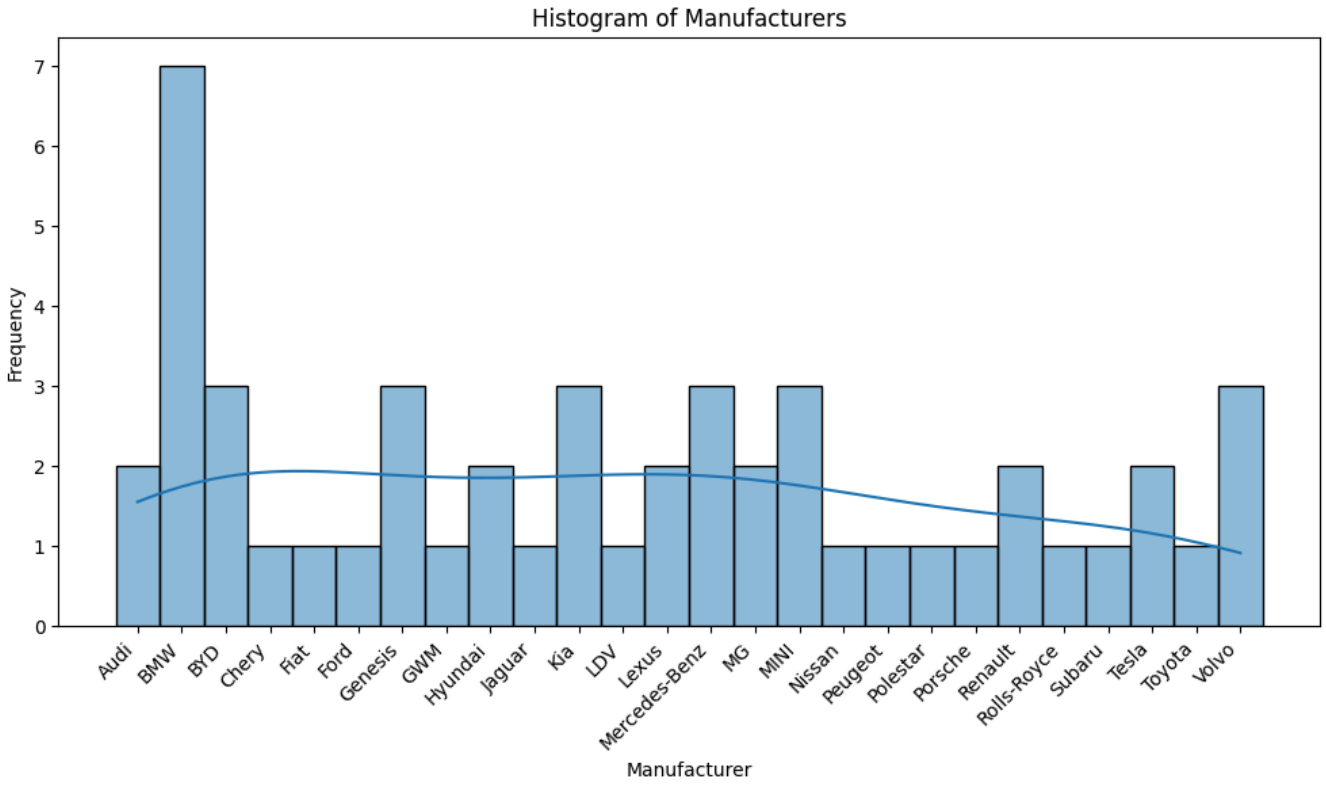
Review Distribution of Vehicle Ranges

```
# prompt: Plot a histogram of RANGE (km)

# Assuming df2 is already loaded and processed as in the previous code.

plt.figure(figsize=(10, 6))
sns.histplot(df2['RANGE (km)'], bins=30, kde=True)
plt.title("Histogram of RANGE (km)")
plt.xlabel("Range (km)")
plt.ylabel("Frequency")
plt.show()
```



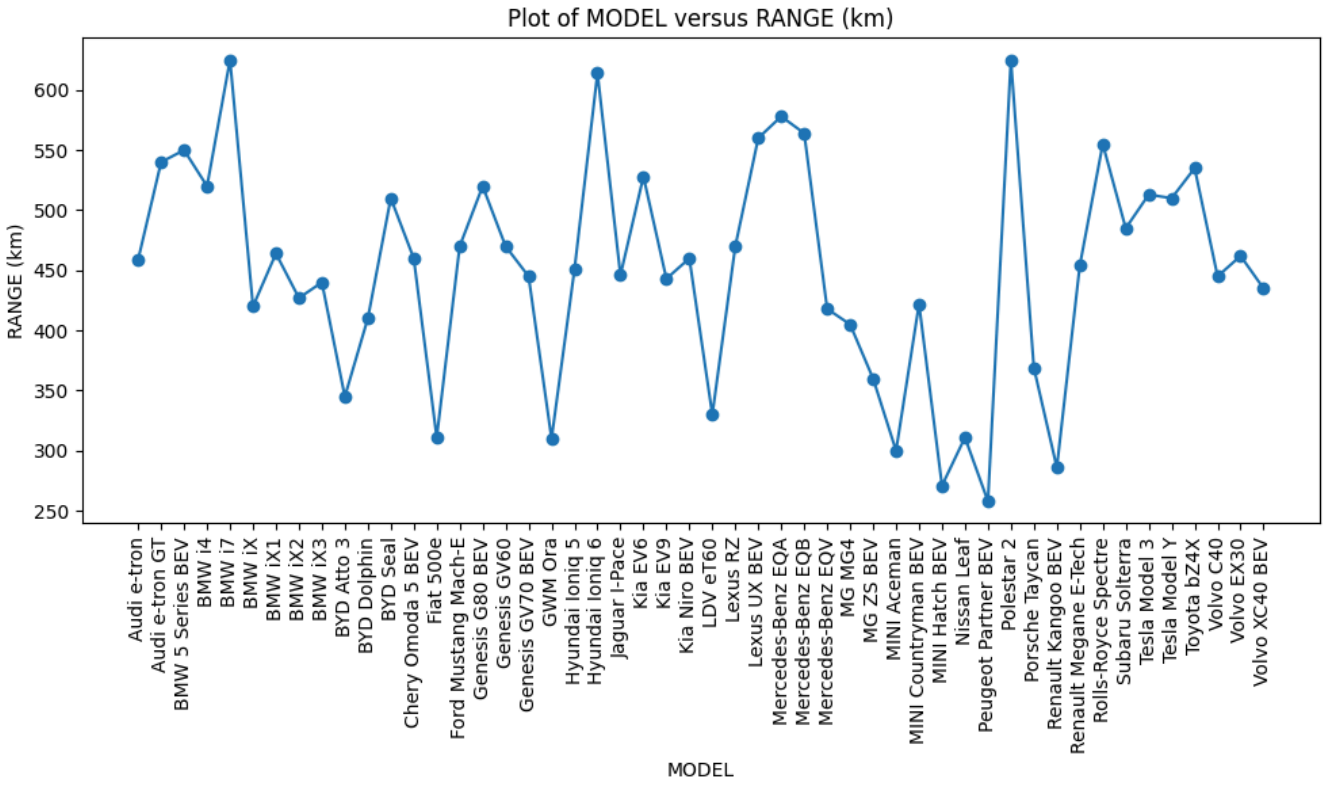
Review of manufacturers & models

```
# prompt: Plot a histogram of Manufacturer

import matplotlib.pyplot as plt
import seaborn as sns

# Assuming df2 is already loaded and processed as in the previous code.

plt.figure(figsize=(10, 6))
sns.histplot(df2['Manufacturer'], bins=30, kde=True)
plt.title("Histogram of Manufacturers")
plt.xlabel("Manufacturer")
plt.ylabel("Frequency")
plt.xticks(rotation=45, ha='right')  # Rotate x-axis labels for readability
plt.tight_layout()  # Adjust layout to prevent labels from overlapping
plt.show()
```



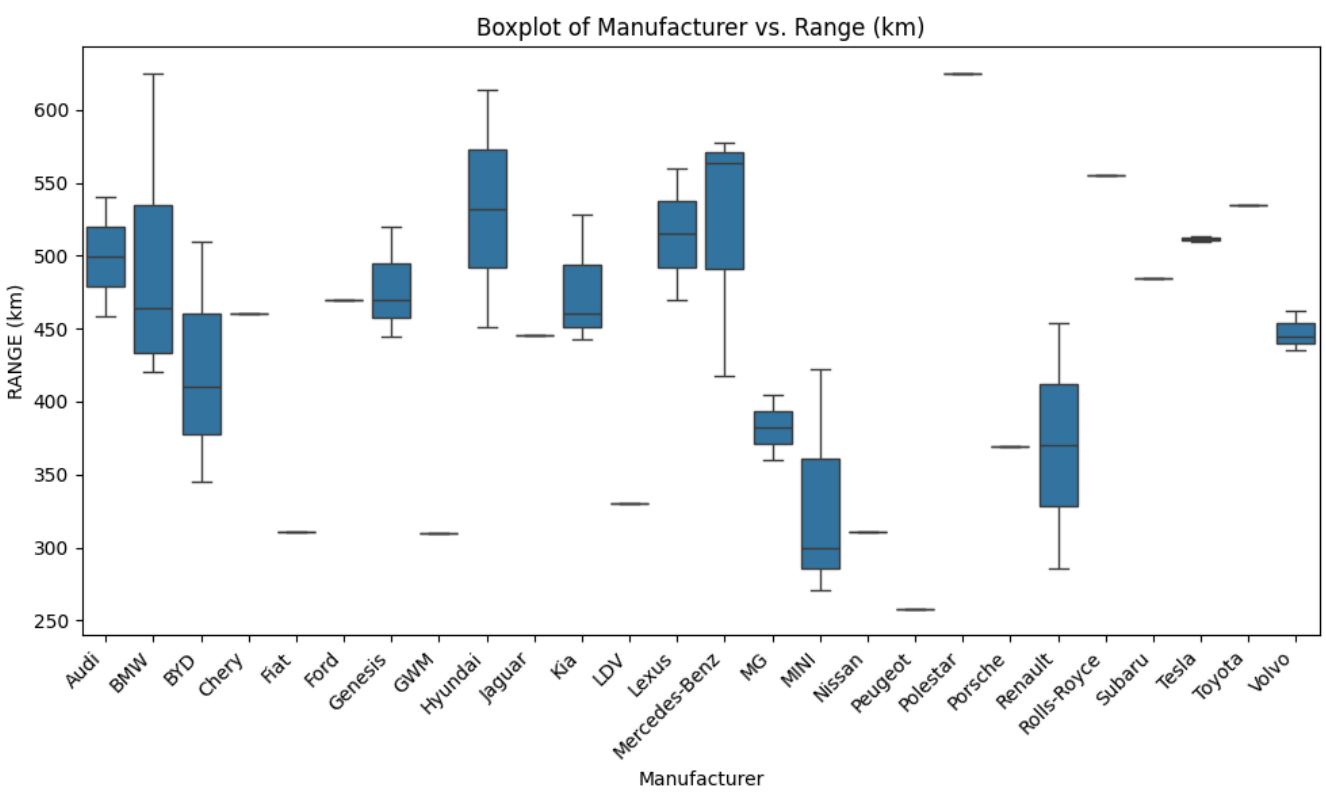
Review of Vehicle Models Versus Range

```
# prompt: Plot of   MODEL versus RANGE (km)

# Assuming df2 is already loaded and processed as in the previous code.

plt.figure(figsize=(10, 6))
plt.plot(df2['MODEL'], df2['RANGE (km)'], marker='o', linestyle='-')  # Create the plot
plt.xlabel("MODEL")
plt.ylabel("RANGE (km)")
plt.title("Plot of MODEL versus RANGE (km)")
plt.xticks(rotation=90)  # Rotate x-axis labels for better readability
plt.tight_layout()  # Adjust layout to prevent labels from overlapping
plt.show()
```



Boxplot of Vehicle Manufacturers versus range

```
# prompt: Plot of  boxplot Manufacturer versus RANGE (km)

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.boxplot(x='Manufacturer', y='RANGE (km)', data=df2)
plt.title('Boxplot of Manufacturer vs. Range (km)')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```

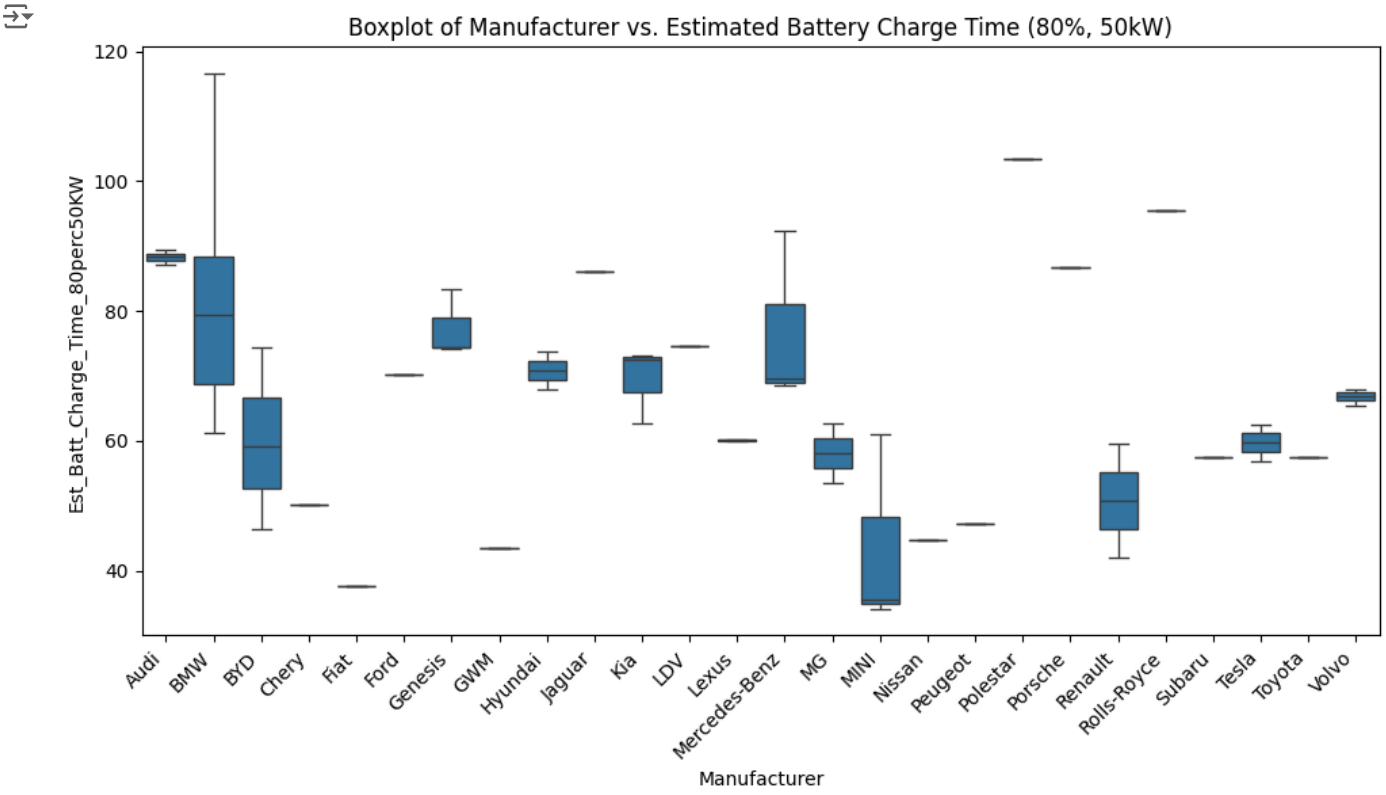


Create an estimate of 80percent battery charge using a 50KW charger as a default value

```
# prompt: Create a new Est_Batt_Charge_Time_80perc50KW column by multiplying Battery_capacity_kW by 1/50 by 60 by 0.7
# Average is from 10% to 80%, 50kw is the most frequency station charger size.

df2['Est_Batt_Charge_Time_80perc50KW'] = df2['Battery_capacity_kW'] * (1/50) * 60 * 0.7
```

Box Plot of Estimated 80 percent vehicle charge with a 50KW Charger

```
# prompt: Plot of  boxplot Manufacturer versus Est_Batt_Charge_Time_80perc50KW

plt.figure(figsize=(10, 6))
sns.boxplot(x='Manufacturer', y='Est_Batt_Charge_Time_80perc50KW', data=df2)
plt.title('Boxplot of Manufacturer vs. Estimated Battery Charge Time (80%, 50kW)')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



Set Default Vehicle Year to 2024 & Vechicle Connector to CCS

```
# prompt: Add a columns for VehicleYear = 2024 and ConnectorType=CCS

# Assuming df2 is already loaded and processed as in the previous code.

# Add new columns
df2['VehicleYear'] = 2024
df2['ConnectorType'] = 'CCS'

# Display the updated DataFrame to verify the changes
df2.head()
```

	VEHICLE TYPE	FUEL TYPE	MODEL	VARIANT DETAILS	LISTED PRICE ($AUD)	FAST CHARGE TIME (minutes)	ANCAP RATING	RANGE (km)	ENERGY CONSUMPTION	Manufacturer	Battery_capacity_kW	Est_I
0	Large SUV	BEV	Audi e-tron	Audi e-tron 55 Auto quattro	NaN	85 mins (5%-80% charge, 50kW charger)	5 star, 2019	459.0	23.2	Audi	106.488	
1	Large Car	BEV	Audi e-tron GT	2024 Audi e-tron GT Auto e-quattro MY24	181784	23 mins (5%-80% charge, 270kW charger)	Unrated	540.0	19.2	Audi	103.680	
2	Large Car	BEV	BMW 5 Series BEV	2024 BMW i5 eDrive40 M Sport G60 Auto	155900	30 mins (10%-80% charge, 205kW charger)	5 star, 2023	550.0	16.5	BMW	90.750	
3	Medium Car	BEV	BMW i4	2024 BMW i4 eDrive35 M Sport G26 Auto	85900	31 mins (10%-80% charge, 205kW charger)	4 star, 2022	520.0	22.2	BMW	115.440	
4	Large Car	BEV	BMW i7	2024 BMW i7 xDrive60 M Sport G70 Auto AWD	306900	34 mins (10%-80% charge, 195kW charger)	Unrated	625.0	22.2	BMW	138.750	

Next steps: Generate code with df2 | View recommended plots | New interactive sheet

Total Number of unique Vehicles

```
# prompt: # count number of unique MODEL in df2

# Assuming df2 is already loaded and processed as in the previous code.

unique_model_count = df2['MODEL'].nunique()
print(f"The number of unique models is: {unique_model_count}")
```

```
The number of unique models is: 50
```

Extension of Vehicle Dataset for Number of Users

```
# prompt: randomly sample the Manufacturer  = Tesla and generate 19 additional rows in df2

# Assuming df2 is already loaded and processed as in the previous code.

# Sample rows where Manufacturer is Tesla
tesla_df = df2[df2['Manufacturer'] == 'Tesla']

# Randomly sample 30 rows from the Tesla DataFrame
sampled_tesla_df = tesla_df.sample(n=19, replace=True)

# Concatenate the sampled rows with the original df2
df2 = pd.concat([df2, sampled_tesla_df], ignore_index=True)
```

```
# prompt: randomly sample the Manufacturer  = BYD and generate 8 additional rows in df2

# Assuming df2 is already loaded and processed as in the previous code.
```

```
# Sample rows where Manufacturer is BYD
byd_df = df2[df2['Manufacturer'] == 'BYD']

# Randomly sample 8 rows from the BYD DataFrame, with replacement to allow duplicates
sampled_byd_df = byd_df.sample(n=8, replace=True)

# Concatenate the sampled rows with the original df2
df2 = pd.concat([df2, sampled_byd_df], ignore_index=True)
```

```
# prompt: randomly sample the Manufacturer  = MG and generate 8 additional rows in df2

# Assuming df2 is already loaded and processed as in the previous code.

# Sample rows where Manufacturer is MG
mg_df = df2[df2['Manufacturer'] == 'MG']

# Randomly sample 8 rows from the MG DataFrame, with replacement to allow duplicates
sampled_mg_df = mg_df.sample(n=8, replace=True)

# Concatenate the sampled rows with the original df2
df2 = pd.concat([df2, sampled_mg_df], ignore_index=True)
```

```
# prompt: print the length of df2

print(len(df2))
```

```
85
```

```
# prompt: add the df_userID as a new column in df2

# Assuming df2 and df_userID are already defined as in your provided code.

# Check if 'userId' column exists in df2. If not, create it.
if 'userId' not in df2.columns:
    df2['userId'] = 0  # Initialize with a default value, or another appropriate value


# Reset the index of df_userID if it's not already a simple numerical index.
df_userID = df_userID.reset_index(drop=True)

# Ensure that df_userID and df2 have compatible lengths for merging.
# If df_userID has more rows than df2, trim df_userID.
min_len = min(len(df_userID), len(df2))
df_userID = df_userID.head(min_len)

# Assign the 'userId' column from df_userID to the new column in df2.
df2['userId'] = df_userID['userId'].values
```

```
df2.head()
```

	VEHICLE TYPE	FUEL TYPE	MODEL	VARIANT DETAILS	LISTED PRICE ($AUD)	FAST CHARGE TIME (minutes)	ANCAP RATING	RANGE (km)	ENERGY CONSUMPTION	Manufacturer	Battery_capacity_kW	Est_I
0	Large SUV	BEV	Audi e-tron	Audi e-tron 55 Auto quattro	NaN	85 mins (5%-80% charge, 50kW charger)	5 star, 2019	459.0	23.2	Audi	106.488	
1	Large Car	BEV	Audi e-tron GT	2024 Audi e-tron GT Auto e-quattro MY24	181784	23 mins (5%-80% charge, 270kW charger)	Unrated	540.0	19.2	Audi	103.680	
2	Large Car	BEV	BMW 5 Series BEV	2024 BMW i5 eDrive40 M Sport G60 Auto	155900	30 mins (10%-80% charge, 205kW charger)	5 star, 2023	550.0	16.5	BMW	90.750	
3	Medium Car	BEV	BMW i4	2024 BMW i4 eDrive35 M Sport G26 Auto	85900	31 mins (10%-80% charge, 205kW charger)	4 star, 2022	520.0	22.2	BMW	115.440	
4	Large Car	BEV	BMW i7	2024 BMW i7 xDrive60 M Sport G70 Auto AWD	306900	34 mins (10%-80% charge, 195kW charger)	Unrated	625.0	22.2	BMW	138.750	

Next steps: Generate code with df2 | View recommended plots | New interactive sheet

```
# prompt: delete alphanumeric_code column from df2

# Assuming df2 is already defined as in your provided code.

# Delete the 'alphanumeric_code' column if it exists
if 'alphanumeric_code' in df2.columns:
    del df2['alphanumeric_code']
```

```
!pip install mimesis --upgrade  # Upgrade mimesis to the latest version
from mimesis import Generic
from mimesis.locales import Locale  # Import Locale instead of AustraliaSpecProvider

# Initialize the Generic object with the Australia locale
generic = Generic(locale=Locale.EN_AU)  # Use Locale.EN_AU for Australia

# Generate a UUID with 4 characters and 4 numbers.
# Mimesis doesn't directly support this combined format.
# Instead, we'll create a helper function

def custom_uuid():
    chars = ''.join(generic.random.choice(generic.text.alphabet()) for _ in range(4)) # Update: use generic.random.choice an
    nums = ''.join(str(generic.random.randint(0, 9)) for _ in range(4)) # Update: use generic.random.randint instead of gene
    return chars + nums

# Example usage:
print(custom_uuid())
```

```
Collecting mimesis
  Downloading mimesis-18.0.0-py3-none-any.whl.metadata (5.7 kB)
Downloading mimesis-18.0.0-py3-none-any.whl (4.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.7/4.7 MB 71.3 MB/s eta 0:00:00
Installing collected packages: mimesis
Successfully installed mimesis-18.0.0
EFMW2868
```

```
# prompt: use the mimesis library to generate a custom uuid of 4 characters and 4 numbers

# Assuming df2 is already defined as in your provided code.

# Add the custom UUIDs as a new column
df2['vehicleID'] = [custom_uuid() for _ in range(len(df2))]

df2.head()
```

	VEHICLE TYPE	FUEL TYPE	MODEL	VARIANT DETAILS	LISTED PRICE ($AUD)	FAST CHARGE TIME (minutes)	ANCAP RATING	RANGE (km)	ENERGY CONSUMPTION	Manufacturer	Battery_capacity_kW	Est_I
0	Large SUV	BEV	Audi e-tron	Audi e-tron 55 Auto quattro	NaN	85 mins (5%-80% charge, 50kW charger)	5 star, 2019	459.0	23.2	Audi	106.488	
1	Large Car	BEV	Audi e-tron GT	2024 Audi e-tron GT Auto e-quattro MY24	181784	23 mins (5%-80% charge, 270kW charger)	Unrated	540.0	19.2	Audi	103.680	
2	Large Car	BEV	BMW 5 Series BEV	2024 BMW i5 eDrive40 M Sport G60 Auto	155900	30 mins (10%-80% charge, 205kW charger)	5 star, 2023	550.0	16.5	BMW	90.750	
3	Medium Car	BEV	BMW i4	2024 BMW i4 eDrive35 M Sport G26 Auto	85900	31 mins (10%-80% charge, 205kW charger)	4 star, 2022	520.0	22.2	BMW	115.440	
4	Large Car	BEV	BMW i7	2024 BMW i7 xDrive60 M Sport G70 Auto AWD	306900	34 mins (10%-80% charge, 195kW charger)	Unrated	625.0	22.2	BMW	138.750	

Next steps: Generate code with df2 | View recommended plots | New interactive sheet

UPDATE OF CHARGING SESSION times with EV vehicle Data & Electricity Prices

```
#load dataset if not already loaded
#df= pd.read_csv('station_data_dataverse.csv')

df.tail(2)
```

	sessionId	kwhTotal	dollars	created	ended	startTime	endTime	chargeTimeHrs	weekday	platform	distance	u
3393	5181950	0.0	0.0	0015-09-23 11:02:55	0015-09-23 13:03:05	11	13	2.002778	Wed	ios	NaN	244
3394	7661889	0.0	0.0	0015-09-28 11:01:45	0015-09-28 11:03:06	11	11	0.022500	Mon	ios	NaN	244

Correction to date time in the Charging Session Data to Add wholesale electricity price at that data

```
# prompt: store the first 4 characters of df[created] in a dataframe and print the unique value in this dataframe to determi
```

```
# Assuming 'df' is your DataFrame and it has a 'created' column.
# Extract the first 4 characters of the 'created' column.  Handle potential errors gracefully.
try:
    df['created_prefix'] = df['created'].astype(str).str[:4]
    unique_prefixes = df['created_prefix'].unique()
    print(unique_prefixes)
except KeyError:
    print("Error: 'created' column not found in the DataFrame.")
except AttributeError:
    print("Error: 'created' column does not have string-like values.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
['0015' '0014']
```

```
# prompt: change the first character of df[created] to 2 ( changing the year to 2014 or 2015)

# Assuming 'df' is your DataFrame and it has a 'created' column.
# Change the first character of df['created'] to '2' if it's a string.
try:
    df['created'] = df['created'].astype(str).str.replace(r'^.', '2', regex=True)
except KeyError:
    print("Error: 'created' column not found in the DataFrame.")
except AttributeError:
    print("Error: 'created' column does not have string-like values.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
df.head(2)
```

	sessionId	kwhTotal	dollars	created	ended	startTime	endTime	chargeTimeHrs	weekday	platform	distance	user
0	4926737	23.68	0.5	2015-10-03 07:18:43	0015-10-03 11:25:07	7	11	4.106667	Sat	android	NaN	789081
1	3738844	22.14	0.0	2015-08-01 05:29:02	0015-08-01 09:00:08	5	9	3.518333	Sat	android	NaN	789081

```
# prompt: convert the df[created] to datetime

# Assuming 'df' is your DataFrame and it has a 'created' column.
# Convert the 'created' column to datetime objects. Handle potential errors gracefully.
try:
    df['created'] = pd.to_datetime(df['created'], errors='coerce')
except KeyError:
    print("Error: 'created' column not found in the DataFrame.")
except ValueError:
    print("Error: Could not convert 'created' column to datetime. Check the format of the values.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

Update the Charging session start times

```
# prompt: extract the hour form df[created] and update df['startTime']

# Assuming 'df' is your DataFrame and it has 'created' and 'startTime' columns.
# Extract the hour from the 'created' column and update the 'startTime' column.

df['startTime'] = df['created'].dt.hour
```

```
df.tail(2)
```

	sessionId	kwhTotal	dollars	created	ended	startTime	endTime	chargeTimeHrs	weekday	platform	distance	u
3393	5181950	0.0	0.0	2015-09-23 11:02:55	0015-09-23 13:03:05	11	13	2.002778	Wed	ios	NaN	244
3394	7661889	0.0	0.0	2015-09-28 11:01:45	0015-09-28 11:03:06	11	11	0.022500	Mon	ios	NaN	244

Advance the year to align with electricity grid data of 2022 & 2023 for Victora Energy Demand , noting the alignmnet to a weekly demand profile

```
# prompt: advance the days and year of df[created] by 8 years adn 3 days

# Assuming 'df' is your DataFrame and it has a 'created' column of datetime objects.
# Advance the 'created' column by 8 years and 3 days.

try:
    df['created'] = df['created'] + pd.DateOffset(years=8, days=4)
except KeyError:
    print("Error: 'created' column not found in the DataFrame.")
except TypeError:
    print("Error: 'created' column is not of datetime type.")
except Exception as e:
    print(f"An unexpected error occurred: {e}")
```

```
# prompt: print the min and max of df[created]

print(f"Min of df['created']: {df['created'].min()}")
print(f"Max of df['created']: {df['created'].max()}")
```

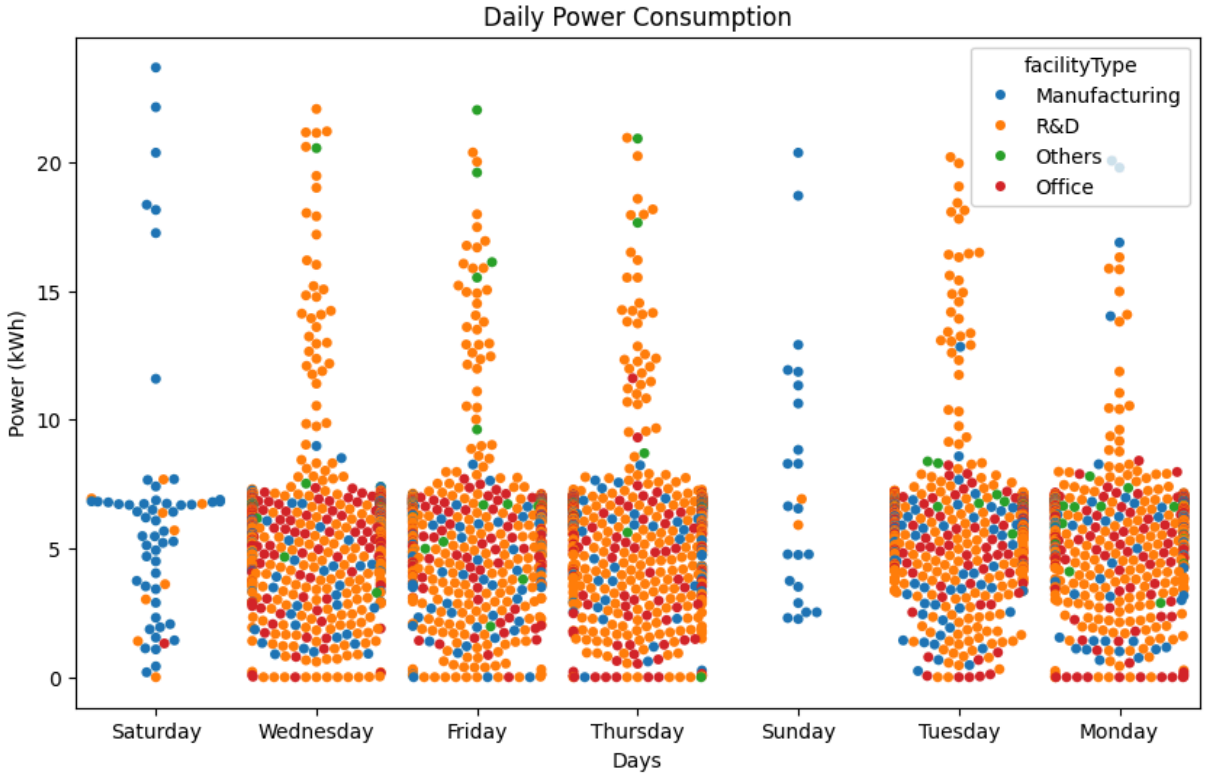
```
Min of df['created']: 2022-11-22 15:01:17
Max of df['created']: 2023-10-08 12:44:59
```

Extract and check the weekday distribution with Sat/Sun having the lowest Charging sessions.

```
# prompt: extract the weekday form df[created] and update df['weekday']

# Assuming 'df' is your DataFrame and it has a 'created' column of datetime objects.
# Extract the weekday from the 'created' column and create a new 'weekday' column.

df['weekday'] = df['created'].dt.day_name()
```

```
#convert facilityType to Catagorical data
df['facilityType'] = df['facilityType'].replace([1,2,3,4],['Manufacturing','Office','R&D','Others'])

#visualize daily power consumption
plt.figure(figsize=(10,6))
sns.swarmplot(data=df, y='kwhTotal', x='weekday', hue='facilityType')
plt.title('Daily Power Consumption')
plt.xlabel('Days')
plt.ylabel('Power (kWh)')
plt.show()
```

```
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
60.6% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
53.9% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
60.7% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
58.4% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
57.6% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
6.5% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
62.0% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
56.1% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
62.3% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
59.7% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
/usr/local/lib/python3.10/dist-packages/seaborn/categorical.py:3399: UserWarning:
58.8% of the points cannot be placed; you may want to decrease the size of the markers or use stripplot.
```




```
df.head()
```

	sessionId	kwhTotal	dollars	created	ended	startTime	endTime	chargeTimeHrs	weekday	platform	distance	us
0	4926737	23.68	0.5	2023-10-07 07:18:43	0015-10-03 11:25:07	7	11	4.106667	Saturday	android	NaN	789(
1	3738844	22.14	0.0	2023-08-05 05:29:02	0015-08-01 09:00:08	5	9	3.518333	Saturday	android	NaN	789(
2	2682332	22.07	0.0	2023-09-06 13:43:27	0015-09-02 17:38:10	13	17	3.911944	Wednesday	android	NaN	789(
3	9025610	22.03	0.0	2023-06-02 16:55:35	0015-05-29 20:54:06	16	20	3.975278	Friday	android	NaN	789(
4	4473237	21.20	0.0	2023-10-04 16:52:49	0015-09-30 20:42:06	16	20	3.821389	Wednesday	android	NaN	789(

For each Charging session, using the UserID, extract the vehicle Battery Capacity, Time Estimate to charge to 80% using a 50kW charger, and replace the Charging session KWTotal to 80% of Battery Capacity

```
# prompt: for each df[userId] find the correspond df2[userId], df2[Battery_capacity_kW] and df2[Est_Batt_Charge_Time_80perc5

# Assuming df and df2 are your DataFrames.

for index, row in df.iterrows():
    user_id = row['userId']
    matching_rows = df2[df2['userId'] == user_id]

    if not matching_rows.empty:
        # Assuming there's only one match per user ID in df2.  If not, adjust accordingly.
        matching_row = matching_rows.iloc[0]

        df.loc[index, 'kwhTotal'] = 0.8 * matching_row['Battery_capacity_kW']
        df.loc[index, 'chargeTimeHrs'] = matching_row['Est_Batt_Charge_Time_80perc50KW']
```

```
# prompt: print any rows of df[kwhTotal] with 0 or NaN

# Assuming df is already loaded and processed as in your provided code.

# Print rows where 'kwhTotal' is 0 or NaN
zero_nan_kwh = df[df['kwhTotal'].isin([0, np.nan])]
zero_nan_kwh
```

	sessionId	kwhTotal	dollars	created	ended	startTime	endTime	chargeTimeHrs	weekday	platform	distance	userId	st

Updates the values of KwHr Total and ChargeTime Hours with a random percentage for each vechicle charging session

```
# prompt:  update values of both columns df[kwhTotal ] and df[chargeTimeHrs] with the same random percentage which varies be

# Create a random percentage change between -10% and +10% for each row
random_percentage_changes = np.random.uniform(-0.1, 0.1, size=len(df))

# Apply the percentage change to both columns
df['kwhTotal'] = df['kwhTotal'] * (1 + random_percentage_changes)
df['chargeTimeHrs'] = df['chargeTimeHrs'] * (1 + random_percentage_changes)
```

```
df.head(5)
```

	sessionId	kwhTotal	dollars	created	ended	startTime	endTime	chargeTimeHrs	weekday	platform	distance	u
0	4926737	92.815080	0.5	2023-10-07 07:18:43	0015-10-03 11:25:07	7	11	97.455835	Saturday	android	NaN	789
1	3738844	80.584214	0.0	2023-08-05 05:29:02	0015-08-01 09:00:08	5	9	84.613424	Saturday	android	NaN	789
2	2682332	82.528564	0.0	2023-09-06 13:43:27	0015-09-02 17:38:10	13	17	86.654993	Wednesday	android	NaN	789
3	9025610	84.131013	0.0	2023-06-02 16:55:35	0015-05-29 20:54:06	16	20	88.337564	Friday	android	NaN	789
4	4473237	77.422927	0.0	2023-10-04 16:52:49	0015-09-30 20:42:06	16	20	81.294073	Wednesday	android	NaN	789

```
# prompt: convert df[chargeTimeHrs] to whole integers

# Convert 'chargeTimeHrs' to integers
df['chargeTimeHrs'] = df['chargeTimeHrs'].astype(int)
```

```
# prompt: change df[chargeTimeHrs] to df[chargeTimeMins]

# Rename the 'chargeTimeHrs' column to 'chargeTimeMins'
df = df.rename(columns={'chargeTimeHrs': 'chargeTimeMins'})
```

Calcate the new End Time for charging

```
# prompt: change value of df[chargeTimeHrs] to a timedelta(hours=value)

# Assuming df is already loaded and processed as in your provided code.

# Convert 'chargeTimeHrs' to timedelta objects
df['chargeTimeMins'] = pd.to_timedelta(df['chargeTimeMins'], unit='m')
```

```
# prompt: df[ended] = df[created] + df[chargeTimeMins]

# Assuming df is already loaded and processed as in your provided code.

# Calculate 'ended' column
df['ended'] = df['created'] + df['chargeTimeMins']
```

```
# prompt: update the name of df['chargeTimeMins'] to df[chargeTimeHrs]

# Rename the 'chargeTimeMins' column to 'chargeTimeHrs'
df = df.rename(columns={'chargeTimeMins': 'chargeTimeHrs'})
```

```
# prompt: extract the hour form df[ended] and update df['endTime']

# Assuming 'df' is your DataFrame and it has 'ended' and 'endTime' columns.
# Extract the hour from the 'ended' column and update the 'endTime' column.

df['endTime'] = df['ended'].dt.hour + 1
```

Add the Most popular Charger ConnectionType to each charging session

```
# prompt: creat df[ConnectorType] with an initial value of 'CCS'. Then for each df[userId] find the correspond df2[userId] a

# Create the 'ConnectorType' column in df and initialize it to 'CCS'
df['ConnectorType'] = 'CCS'

# Iterate through rows of df
```

```
for index, row in df.iterrows():
    user_id = row['userId']
    # Find the corresponding row in df2 based on 'userId'
    matching_rows = df2[df2['userId'] == user_id]

    if not matching_rows.empty:
        # Assuming there's only one match per user ID in df2. If not, adjust accordingly.
        connector_type_from_df2 = matching_rows['ConnectorType'].iloc[0]

        # Update the 'ConnectorType' column with the value from df2
        df.loc[index, 'ConnectorType'] = connector_type_from_df2  # Assign directly
```

Add a Default Base Price & Peak Adjustment Price to Charging Session Data

```
df.head(2)
```

	sessionId	kwhTotal	dollars	created	ended	startTime	endTime	chargeTimeHrs	weekday	platform	distance	use
0	4926737	92.815080	0.5	2023-10-07 07:18:43	2023-10-07 08:55:43	7	9	0 days 01:37:00	Saturday	android	NaN	78908
1	3738844	80.584214	0.0	2023-08-05 05:29:02	2023-08-05 06:53:02	5	7	0 days 01:24:00	Saturday	android	NaN	78908

```
# prompt: add a new column name df2[BasePrice] and set df2[BasePrice] = 0.45

# Assuming df2 is already loaded and processed as in the previous code.

BasePrice = 0.55

df['BasePrice'] = BasePrice
```

```
# prompt: add a new column named df2[PeakPriceAdjustment] and set df2[PeakPriceAdjustment] = 0.5
#Can be varied to adjust for peak periods.

PeakPriceAdjustment = 0.0

df['PeakPriceAdjustment'] = PeakPriceAdjustment
```

INCORPORATE VICTORIA WHOLESALE ELECTRICITY PRICES

Victoria Energy 5 min Demand Proile

This work focusses on the locaised region of Victoria Energy Demand to better understand the dynamics of EV Impact & Deployment scenarios, Demand & Charging Infrastructure, Oil Displacement and Emissions Reduction.

it is important for electricity generators to understand the changing dynamics of the electricity demand.

The Data is sourced from the following link:

https://aemo.com.au/energy-systems/electricity/national-electricity-market-nem/data-nem/aggregated-data

```
#import packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
sns.set_style('dark')

import statsmodels.api as sm
from sklearn.metrics import mean_absolute_error
#from features_preprocessing import transform_to_windows, rename_cols
```

```
!tar -xzvf Energy_Demand.tar.gz
```

```
Energy_Demand/
Energy_Demand/PRICE_AND_DEMAND_202201_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202202_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202203_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202204_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202205_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202206_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202207_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202208_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202209_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202210_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202211_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202212_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202301_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202302_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202303_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202304_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202305_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202306_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202307_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202308_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202309_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202310_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202311_VIC1.csv
Energy_Demand/PRICE_AND_DEMAND_202312_VIC1.csv
```

```
import pandas as pd
import glob

# Get a list of all CSV files in a directory
csv_files = glob.glob('/content/Energy_Demand/*.csv')
```

```
# Create an empty dataframe to store the combined data
combined_df = pd.DataFrame()
```

```
# Loop through each CSV file and append its contents to the combined dataframe
for csv_file in csv_files:
    df6 = pd.read_csv(csv_file)
    combined_df = pd.concat([combined_df, df6])
```

```
loaded_energy_data = combined_df.copy()
```

```
loaded_energy_data.tail(5)
```

	REGION	SETTLEMENTDATE	TOTALDEMAND	RRP	PERIODTYPE
8635	VIC1	2022/11/30 23:40:00	4486.28	113.12	TRADE
8636	VIC1	2022/11/30 23:45:00	4444.92	113.03	TRADE
8637	VIC1	2022/11/30 23:50:00	4456.04	113.99	TRADE
8638	VIC1	2022/11/30 23:55:00	4372.91	105.36	TRADE
8639	VIC1	2022/12/01 00:00:00	4340.46	104.68	TRADE

Drop Irrelevant columns, and data with in correct settlements dates

```
# prompt: drop columns REGION TOTALDEMAND and PERIODTYPE from loaded_energy_data

# Assuming loaded_energy_data is your DataFrame.

# Drop the specified columns if they exist
columns_to_drop = ['REGION', 'TOTALDEMAND', 'PERIODTYPE']
for col in columns_to_drop:
    if col in loaded_energy_data.columns:
        loaded_energy_data = loaded_energy_data.drop(col, axis=1)
```

```
# prompt: convert loaded_energy_data[SETTLEMENTDATE] to a string

# Assuming loaded_energy_data is already loaded as shown in your provided code.
```

```
# Convert the 'SETTLEMENTDATE' column to string type.
loaded_energy_data['SETTLEMENTDATE'] = loaded_energy_data['SETTLEMENTDATE'].astype(str)
```

```
# prompt: print the unique lengths of loaded_energy_data['SETTLEMENTDATE']

unique_lengths = loaded_energy_data['SETTLEMENTDATE'].str.len().unique()
unique_lengths
```

```
array([19, 14, 15, 16])
```

```
# prompt: remove rows in loaded_energy_data where  lengths of loaded_energy_data['SETTLEMENTDATE'] is less than 19

# Assuming loaded_energy_data is already loaded and processed as in your provided code.

# Remove rows where the length of 'SETTLEMENTDATE' is less than 19
loaded_energy_data = loaded_energy_data[loaded_energy_data['SETTLEMENTDATE'].str.len() >= 19]
```

Convert & Set Date Time index for interpolation

```
# prompt: convert loaded_energy_data[SETTLEMENTDATE] to date time

# Convert 'SETTLEMENTDATE' to datetime objects
# Specify the format to match the data format in the column.
loaded_energy_data['SETTLEMENTDATE'] = pd.to_datetime(loaded_energy_data['SETTLEMENTDATE'], format='%Y/%m/%d %H:%M:%S')
```

```
<ipython-input-146-324790f73531>:5: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-
```

```
loaded_energy_data
```

	SETTLEMENTDATE	RRP
0	2022-08-01 00:05:00	8.94
1	2022-08-01 00:10:00	8.94
2	2022-08-01 00:15:00	8.94
3	2022-08-01 00:20:00	8.94
4	2022-08-01 00:25:00	8.94
...	...	...
8635	2022-11-30 23:40:00	113.12
8636	2022-11-30 23:45:00	113.03
8637	2022-11-30 23:50:00	113.99
8638	2022-11-30 23:55:00	105.36
8639	2022-12-01 00:00:00	104.68

201312 rows × 2 columns

```
# prompt: Set the loaded_energy_data index to SETTLEMENTDATE

# Assuming loaded_energy_data is already loaded and processed as in your provided code.

# Set 'SETTLEMENTDATE' as the index
loaded_energy_data = loaded_energy_data.set_index('SETTLEMENTDATE')
```

Resample & Interpolate the spot electricity price

```
# prompt: resample the loaded_energy_data index to seconds , and user linear interpolation to find the resample value of RRP

# Resample the index to seconds and use linear interpolation for RRP
loaded_energy_data = loaded_energy_data.resample('S').interpolate(method='linear')
```

```
<ipython-input-149-81d901ff0cf3>:4: FutureWarning:

'S' is deprecated and will be removed in a future version, please use 's' instead.
```

```
loaded_energy_data
```

SETTLEMENTDATE	RRP
2022-02-01 00:05:00	120.960000
2022-02-01 00:05:01	121.544967
2022-02-01 00:05:02	122.129933
2022-02-01 00:05:03	122.714900
2022-02-01 00:05:04	123.299867
...	...
2023-12-31 23:59:56	51.430000
2023-12-31 23:59:57	51.430000
2023-12-31 23:59:58	51.430000
2023-12-31 23:59:59	51.430000
2024-01-01 00:00:00	51.430000

60393301 rows × 1 columns

Redcuce the resampled dataset size ****

```
# prompt: if loaded_energy_data[SETTLEMENTDATE] is not in df[created] the delete that row in loaded_energy_data
ERROR: Delete or comment if cell needs to be run to generate CSV, which 30mins+
# Assuming 'df' and 'loaded_energy_data' are already defined DataFrames.
# Assuming 'created' column exists in 'df' and 'SETTLEMENTDATE' in 'loaded_energy_data'.

# Convert 'created' column to datetime objects if it's not already.
if not pd.api.types.is_datetime64_any_dtype(df['created']):
    try:
        df['created'] = pd.to_datetime(df['created'])
    except (KeyError, ValueError, TypeError):
        print("Error converting 'created' to datetime.")
        # Handle the error appropriately, e.g., skip the operation or raise an exception

# Convert 'SETTLEMENTDATE' column to datetime objects if it's not already.
# Access the index of loaded_energy_data instead of columns
if not pd.api.types.is_datetime64_any_dtype(loaded_energy_data.index):
  try:
    loaded_energy_data.index = pd.to_datetime(loaded_energy_data.index)  # Convert the index to datetime
  except (KeyError, ValueError, TypeError):
    print("Error converting 'SETTLEMENTDATE' to datetime.")
    # Handle the error appropriately

# Create a set of 'created' dates for faster lookup
created_dates = set(df['created'].dt.to_pydatetime())

# Iterate through loaded_energy_data and delete rows where SETTLEMENTDATE is not in df['created']
index_to_delete = []
for index, row in loaded_energy_data.iterrows():
    if index not in created_dates:  # Check against the index now
        index_to_delete.append(index)

loaded_energy_data = loaded_energy_data.drop(index_to_delete)
```

```
  File "<ipython-input-151-ef67ebe15890>", line 2
    ERROR: Delete or comment if cell needs to be run to generate CSV, which 30mins+
                                                                            ^
SyntaxError: invalid decimal literal
```

Next steps: Fix error

Write and Read loaded_energy_data CSV

```
len (loaded_energy_data), len(df['created'])
```

```
# prompt: write the dataframe loaded_energy_data to a csv file

# Assuming loaded_energy_data is already defined and processed.
loaded_energy_data.to_csv('loaded_energy_data.csv', index=True)
```

```
  # prompt: load loaded_energy_data.csv into a dataframe name loaded_energy_data

  loaded_energy_data = pd.read_csv('loaded_energy_data.csv', index_col=0)
```

Review the Min and Max RRP for Wholesale Electricity

```
# prompt: print the minimum and maximum of loaded_energy_data[RRP]

print(f"Minimum RRP: {loaded_energy_data['RRP'].min()}")
print(f"Maximum RRP: {loaded_energy_data['RRP'].max()}")
```

```
Minimum RRP: -994.492
Maximum RRP: 9966.766
```

Convert RRP per MW to RRP per KW

```
# prompt: Change the column name from RRP to RRPperMW in the dataframe loaded_energy_data

# Rename the 'RRP' column to 'RRPperMW'
loaded_energy_data = loaded_energy_data.rename(columns={'RRP': 'RRPperMW'})
```

```
# prompt: add a new column loaded_energy_data[RRPperKW] = loaded_energy_data[RRPperMW] /1000

# Assuming loaded_energy_data is already defined and processed.

loaded_energy_data['RRPperKW'] = loaded_energy_data['RRPperMW'] / 1000
```

```
loaded_energy_data.tail(2)
```

SETTLEMENTDATE	RRPperMW	RRPperKW
2023-10-07 15:49:09	22.7279	0.022728
2023-10-08 12:44:59	-20.0155	-0.020015

```
len(loaded_energy_data)
```

```
3393
```

```
# prompt: convert loaded_energy_data[SETTLEMENT] to data time and the dataframe index

# Assuming loaded_energy_data is already loaded and processed as in your provided code.

# Convert the index to datetime if it's not already
if not pd.api.types.is_datetime64_any_dtype(loaded_energy_data.index):
    loaded_energy_data.index = pd.to_datetime(loaded_energy_data.index)
```

```
# Now the index is a DateTimeIndex, so you can use it directly in the DataFrame
```

Calculate RRPperKW24hrs which is the daily rolling mean of loaded_energy_data[RRPperKW] over the index of loaded_energy_data

Also Review the Min & Max Value

```
# prompt: add a new column loaded_energy_data[RRPperKW24hrs] which is the daily rolling mean of loaded_energy_data[RRPperKW]

# Calculate the daily rolling mean of 'RRPperKW'
loaded_energy_data['RRPperKW24hrs'] = loaded_energy_data['RRPperKW'].rolling(window='24H').mean()
```

```
<ipython-input-159-e8d588b49359>:4: FutureWarning:

'H' is deprecated and will be removed in a future version, please use 'h' instead.
```

```
loaded_energy_data.head()
```

SETTLEMENTDATE	RRPperMW	RRPperKW	RRPperKW24hrs
2022-11-22 15:01:17	-38.000000	-0.038000	-0.038000
2022-11-22 15:40:26	-17.119200	-0.017119	-0.027560
2022-11-23 17:40:26	149.500000	0.149500	0.149500
2022-11-23 19:01:41	283.847633	0.283848	0.216674
2022-11-24 19:20:45	299.500000	0.299500	0.299500

Next steps: Generate code with `loaded_energy_data` View recommended plots New interactive sheet

```
# prompt: print the minimum and maximum of loaded_energy_data[RRPperKW24hrs]

print(f"Minimum RRPperKW24hrs: {loaded_energy_data['RRPperKW24hrs'].min()}")
print(f"Maximum RRPperKW24hrs: {loaded_energy_data['RRPperKW24hrs'].max()}")
```

```
Minimum RRPperKW24hrs: -0.44588132499999994
Maximum RRPperKW24hrs: 2.131064826666667
```

Add the RRPperKW & RRPperKW24Hrs to Charging Session Data

```
# prompt: for each df[created] find the corresponding index in loaded_energy_data and the loaded_energy_data['RRPperKW'] , a

# Assuming df and loaded_energy_data are already defined DataFrames.

df['RRPperKW'] = 0.0  # Initialize the new column

for index, row in df.iterrows():
    created_time = row['created']
    try:
        # Find the closest time in loaded_energy_data
        closest_time = min(loaded_energy_data.index, key=lambda x: abs(x - created_time))

        # Get the RRPperKW value at the closest time
        rpperkw_value = loaded_energy_data.loc[closest_time, 'RRPperKW']

        # Assign the RRPperKW value to the new column in df
        df.loc[index, 'RRPperKW'] = rpperkw_value
    except KeyError:
        print(f"Warning: No RRPperKW value found for created time {created_time}.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
```

```
# prompt: for each df[created] find the corresponding index in loaded_energy_data and the loaded_energy_data['RRPperKW24hrs'
```

```
# Assuming df and loaded_energy_data are already defined DataFrames.
df['RRPperKW24hrs'] = 0.0  # Initialize the new column

for index, row in df.iterrows():
    created_time = row['created']
    try:
        # Find the closest time in loaded_energy_data
        closest_time = min(loaded_energy_data.index, key=lambda x: abs(x - created_time))

        # Get the RRPperKW24hrs value at the closest time
        rpperkw24hrs_value = loaded_energy_data.loc[closest_time, 'RRPperKW24hrs']

        # Assign the RRPperKW24hrs value to the new column in df
        df.loc[index, 'RRPperKW24hrs'] = rpperkw24hrs_value
    except KeyError:
        print(f"Warning: No RRPperKW24hrs value found for created time {created_time}.")
    except Exception as e:
        print(f"An unexpected error occurred: {e}")
```

```
# prompt: add a new column df[CustPricePerKW] = df[BasePrice] + df[PeakPriceAdjustment] + df[RRPperKW24hrs]

df['CustPricePerKW'] = df['BasePrice'] + df['PeakPriceAdjustment'] + df['RRPperKW24hrs']
```

```
# prompt: Check df['CustPricePerKW'] for Nan values and print the minimum and maximum

# Check for NaN values in 'CustPricePerKW'
nan_values = df['CustPricePerKW'].isna().sum()
print(f"Number of NaN values in 'CustPricePerKW': {nan_values}")

# Print the minimum and maximum values, handling potential NaN values
if nan_values > 0:
    min_cust_price = df['CustPricePerKW'].min(skipna=True)  # Skip NaN values for min
    max_cust_price = df['CustPricePerKW'].max(skipna=True)  # Skip NaN values for max
    print(f"Minimum 'CustPricePerKW' (excluding NaN): {min_cust_price}")
    print(f"Maximum 'CustPricePerKW' (excluding NaN): {max_cust_price}")
else:
    min_cust_price = df['CustPricePerKW'].min()
    max_cust_price = df['CustPricePerKW'].max()
    print(f"Minimum 'CustPricePerKW': {min_cust_price}")
    print(f"Maximum 'CustPricePerKW': {max_cust_price}")
```

```
Number of NaN values in 'CustPricePerKW': 0
Minimum 'CustPricePerKW': 0.1041186750000001
Maximum 'CustPricePerKW': 2.681064826666667
```

```
# prompt: change df[dollars] to df[CustPayAmount]

# Rename the 'dollars' column to 'CustPayAmount' if it exists
if 'dollars' in df.columns:
    df = df.rename(columns={'dollars': 'CustPayAmount'})
```

```
# prompt: df[CustPayAmount] = df[kwhTotal]*df[CustPricePerKW]

df['CustPayAmount'] = df['kwhTotal'] * df['CustPricePerKW']
```

```
# prompt: print the maximium and minimum for df['CustPayAmount']

print(f"Minimum CustPayAmount: {df['CustPayAmount'].min()}")
print(f"Maximum CustPayAmount: {df['CustPayAmount'].max()}")
```

```
Minimum CustPayAmount: 4.295095884134886
Maximum CustPayAmount: 266.2232948850434
```

```
df.head()
```

	sessionId	kwhTotal	CustPayAmount	created	ended	startTime	endTime	chargeTimeHrs	weekday	platform	distanc
0	4926737	92.815080	52.914811	2023-10-07 07:18:43	2023-10-07 08:55:43	7	9	0 days 01:37:00	Saturday	android	Na
1	3738844	80.584214	42.910162	2023-08-05 05:29:02	2023-08-05 06:53:02	5	7	0 days 01:24:00	Saturday	android	Na
2	2682332	82.528564	43.731282	2023-09-06 13:43:27	2023-09-06 15:09:27	13	16	0 days 01:26:00	Wednesday	android	Na
3	9025610	84.131013	59.752299	2023-06-02 16:55:35	2023-06-02 18:23:35	16	19	0 days 01:28:00	Friday	android	Na
4	4473237	77.422927	46.193567	2023-10-04 16:52:49	2023-10-04 18:13:49	16	19	0 days 01:21:00	Wednesday	android	Na

```
# prompt: find the latest date from df[created] and the corresponding value for df[CustPricePerKW]

# Find the latest date in the 'created' column
latest_date = df['created'].max()

# Find the corresponding 'CustPricePerKW' value for the latest date
latest_price = df[df['created'] == latest_date]['CustPricePerKW'].iloc[0]

print(f"Latest Date: {latest_date}")
print(f"Corresponding CustPricePerKW: {latest_price}")
```

```
Latest Date: 2023-10-08 12:44:59
Corresponding CustPricePerKW: 0.5513562000000001
```

Add Random APayment Type to Charging Data

```
# prompt: create a new df[PaymentType] that randonmly selects s payment type from ['American Express', 'Visa', 'MasterCard',

# Create a new column 'PaymentType' with randomly selected payment types
payment_types = ['American Express', 'Visa', 'MasterCard', 'Cash']
df['PaymentType'] = np.random.choice(payment_types, size=len(df))
```

Calculate the Slot Length

```
# prompt: create a new df[SlotLength] = df[endTime] - df[startTime]

# Assuming df is already loaded and processed as in your provided code.

# Calculate 'SlotLength'
df['SlotLength'] = df['endTime'] - df['startTime']
```

Calculate the Booked Timeslots

```
# prompt: create a list of all the integer numbers between  df['startTime'] and df['endTime'] and add them to df[BookingTime

# Create an empty list to store the time slots
df['BookingTimeSlots'] = [[] for _ in range(len(df))] # Initialize as a column of empty lists

# Iterate through the DataFrame
for index, row in df.iterrows():
    start_time = int(row['startTime'])
    end_time = int(row['endTime'])

    # Create a list of integers between start_time and end_time (inclusive)
    time_slots = list(range(start_time, end_time + 1))
```

```
    # Add the time slots to the 'BookingTimeSlots' column
```

Add a Booking Deposit

```
# prompt: add a new column df[BookingDeposit] = 1.00
# A nominal amount > min. Customer transaction.

df['BookingDeposit'] = 1.00
```

Add an Estimate Cost for a Slot length with a 50Kw Charger

```
# prompt: add a new column df[EstCost50KWCharger] = df[SlotLength]*50*(df[BasePrice]+df[PeakPriceAdjustment]

# Assuming df is already loaded and processed as in your provided code.

# Calculate 'EstCost50KWCharger'
df['EstCost50KWCharger'] = df['SlotLength'] * 50 * (df['BasePrice'] + df['PeakPriceAdjustment'])
```

```
df.head()
```

	sessionId	kwhTotal	CustPayAmount	created	ended	startTime	endTime	chargeTimeHrs	weekday	platform	distanc
0	4926737	92.815080	52.914811	2023-10-07 07:18:43	2023-10-07 08:55:43	7	9	0 days 01:37:00	Saturday	android	Na
1	3738844	80.584214	42.910162	2023-08-05 05:29:02	2023-08-05 06:53:02	5	7	0 days 01:24:00	Saturday	android	Na
2	2682332	82.528564	43.731282	2023-09-06 13:43:27	2023-09-06 15:09:27	13	16	0 days 01:26:00	Wednesday	android	Na
3	9025610	84.131013	59.752299	2023-06-02 16:55:35	2023-06-02 18:23:35	16	19	0 days 01:28:00	Friday	android	Na
4	4473237	77.422927	46.193567	2023-10-04 16:52:49	2023-10-04 18:13:49	16	19	0 days 01:21:00	Wednesday	android	Na

Creation of Database Tables

Create EV Database Table CSV

Table EV {

Id varchar [PK, unique]

Manufacturer varchar

Model varchar

VehicleYear varchar

VehicleClass varchar

FuelType varchar

ConnectorType varchar

BatteryCapacityKwh float [ref: - EV.ElectricRangeKm]

EstBattChargeTime80perc50KW float [ref: - EV.BatteryCapacityKwh]

EnergyConsumptionWhkm varchar [ref: - EV.ElectricRangeKm]

ElectricRangeKm float

UserId int

}

```
# prompt: list columns of df2

df2.columns
```

```
Index(['VEHICLE TYPE', 'FUEL TYPE', 'MODEL', 'VARIANT DETAILS',
       'LISTED PRICE ($AUD)', 'FAST CHARGE TIME (minutes)', 'ANCAP RATING',
       'RANGE (km)', 'ENERGY CONSUMPTION', 'Manufacturer',
       'Battery_capacity_kW', 'Est_Batt_Charge_Time_80perc50KW', 'VehicleYear',
       'ConnectorType', 'userId', 'vehicleID'],
      dtype='object')
```

```
# prompt: Copy df2 to a dataframe name ev

ev = df2.copy()
```

```
# prompt: drop ev[VARIANT DETAILS], ev[LISTED PRICE ($AUD)], ev[FAST CHARGE TIME (minutes)], ev[ANCAP RATING]

# Assuming df2 is already loaded and processed as in the previous code.

# Drop specified columns
columns_to_drop = ['VARIANT DETAILS', 'LISTED PRICE ($AUD)', 'FAST CHARGE TIME (minutes)', 'ANCAP RATING']
ev = df2.drop(columns=columns_to_drop, errors='ignore') # Use errors='ignore' to avoid errors if columns don't exist

# Display the updated DataFrame to verify the changes
ev.head(2)
```

	VEHICLE TYPE	FUEL TYPE	MODEL	RANGE (km)	ENERGY CONSUMPTION	Manufacturer	Battery_capacity_kW	Est_Batt_Charge_Time_80perc50KW	VehicleY
0	Large SUV	BEV	Audi e-tron	459.0	23.2	Audi	106.488	89.44992	2
1	Large Car	BEV	Audi e-tron GT	540.0	19.2	Audi	103.680	87.09120	2

Next steps: Generate code with ev | View recommended plots | New interactive sheet

```
# prompt: change ev[vehicleID] to ev[Id] , and ev[userId] to ev[UserId], and ev[Est_Batt_Charge_Time_80perc50KW] to ev[EstBa

ev = ev.rename(columns={
    'vehicleID': 'Id',
    'userId': 'UserId',
    'Est_Batt_Charge_Time_80perc50KW': 'EstBattChargeTime80perc50KW',
    'Battery_capacity_kW': 'BatteryCapacityKwh',
    'ENERGY CONSUMPTION': 'EnergyConsumptionWhkm',
    'RANGE (km)': 'ElectricRangeKm',
    'MODEL': 'Model',
    'VEHICLE TYPE': 'VehicleClass',
    'FUEL TYPE': 'FuelType'
})
```

```
ev.head(2)
```

	VehicleClass	FuelType	Model	ElectricRangeKm	EnergyConsumptionWhkm	Manufacturer	BatteryCapacityKwh	EstBattCharge
0	Large SUV	BEV	Audi e-tron	459.0	23.2	Audi	106.488	
1	Large Car	BEV	Audi e-tron GT	540.0	19.2	Audi	103.680	

Next steps: Generate code with ev | View recommended plots | New interactive sheet

```
# prompt: Display the data types of columns in EV

ev.dtypes
```

	0
VehicleClass	object
FuelType	object
Model	object
ElectricRangeKm	float64
EnergyConsumptionWhkm	float64
Manufacturer	object
BatteryCapacityKwh	float64
EstBattChargeTime80perc50KW	float64
VehicleYear	int64
ConnectorType	object
UserId	Int64
Id	object

dtype: object

```
# prompt: write the dataframe ev to a csv file

ev.to_csv('ev.csv', index=False)
```

Create PurchaseWholesaleElec Database Table CSV

Table PurchaseWholesaleElec {

Id int [PK, unique]

StationId int

SupplyId int

ExpenditureCode varchar

RRPperKW float [ref: > PurchaseWholesaleElec.RRPperKW24hrs]

RRPperKW24hrs float [ref: - PurchaseWholesaleElec.CustPricePerKW]

BasePrice money [ref: - PurchaseWholesaleElec.CustPricePerKW]

PeakPriceAdjustment money [ref: - PurchaseWholesaleElec.CustPricePerKW]

CustPricePerKW money

PurchaseDate datetime

}

```
df.head(2)
```

	sessionId	kwhTotal	CustPayAmount	created	ended	startTime	endTime	chargeTimeHrs	weekday	platform	distance
0	4926737	92.815080	52.914811	2023-10-07 07:18:43	2023-10-07 08:55:43	7	9	0 days 01:37:00	Saturday	android	NaN
1	3738844	80.584214	42.910162	2023-08-05 05:29:02	2023-08-05 06:53:02	5	7	0 days 01:24:00	Saturday	android	NaN

```
# prompt: add column df[SupplyId] with a Default Value of VIC and a column df[ExpenditureCode] with a default value Wholesal

# Add the new columns with default values
df['SupplyId'] = 'VIC'
df['ExpenditureCode'] = 'Wholesale_cost'
```

```
# prompt: add a column df[PurchaseDate] = df[created]

df['PurchaseDate'] = df['created']
```

```
# prompt: copy dataframe df to dataframe PurchaseWholesaleElec

PurchaseWholesaleElec = df.copy()
```

```
PurchaseWholesaleElec.head(2)
```

	sessionId	kwhTotal	CustPayAmount	created	ended	startTime	endTime	chargeTimeHrs	weekday	platform	distance
0	4926737	92.815080	52.914811	2023-10-07 07:18:43	2023-10-07 08:55:43	7	9	0 days 01:37:00	Saturday	android	NaN
1	3738844	80.584214	42.910162	2023-08-05 05:29:02	2023-08-05 06:53:02	5	7	0 days 01:24:00	Saturday	android	NaN

```
# prompt: change PurchaseWholesaleElec[sessionId] to PurchaseWholesaleElec[Id]

# Assuming df and PurchaseWholesaleElec are already defined DataFrames.

# Rename the 'sessionId' column to 'Id' in PurchaseWholesaleElec
if 'sessionId' in PurchaseWholesaleElec.columns:
    PurchaseWholesaleElec = PurchaseWholesaleElec.rename(columns={'sessionId': 'Id'})
    print("Column 'sessionId' renamed to 'Id' in PurchaseWholesaleElec")
else:
    print("Column 'sessionId' not found in PurchaseWholesaleElec. No renaming performed.")

PurchaseWholesaleElec.head(2)
```

```
Column 'sessionId' renamed to 'Id' in PurchaseWholesaleElec
```

	Id	kwhTotal	CustPayAmount	created	ended	startTime	endTime	chargeTimeHrs	weekday	platform	distance	
0	4926737	92.815080	52.914811	2023-10-07 07:18:43	2023-10-07 08:55:43	7	9	0 days 01:37:00	Saturday	android	NaN	7
1	3738844	80.584214	42.910162	2023-08-05 05:29:02	2023-08-05 06:53:02	5	7	0 days 01:24:00	Saturday	android	NaN	7

```
# prompt: From dataframe PurchaseWholesaleElec drop the following columns kwhTotal, CustPayAmount, created, ended, startTime

columns_to_drop = ['kwhTotal', 'CustPayAmount', 'created', 'ended', 'startTime', 'endTime', 'chargeTimeHrs',
                   'weekday' , 'platform', 'distance', 'userId', 'managerVehicle', 'facilityType', 'Mon',
                   'Tues', 'Wed', 'Thurs', 'Fri', 'Sat', 'Sun', 'reportedZip', 'startTime_int', 'created_prefix',
                   'ConnectorType', 'PaymentType', 'SlotLength', 'BookingTimeSlots', 'BookingDeposit',
                   'EstCost50KWCharger', 'locationId']

PurchaseWholesaleElec = PurchaseWholesaleElec.drop(columns=columns_to_drop, errors='ignore')
```

```
# prompt: change PurchaseWholesaleElec[stationId] to PurchaseWholesaleElec[StationId], PurchaseWholesaleElec[locationId] to

# Rename the columns
PurchaseWholesaleElec = PurchaseWholesaleElec.rename(columns={'stationId': 'StationId', 'locationId': 'LocationId'})
```

```
# prompt: List the datatypes of PurchaseWholesaleElec

import pandas as pd
```

```
# Assuming 'df' and 'PurchaseWholesaleElec' are already defined DataFrames as in your provided code.

# Display the data types of columns in PurchaseWholesaleElec
PurchaseWholesaleElec.dtypes
```

	0
Id	int64
StationId	int64
BasePrice	float64
PeakPriceAdjustment	float64
RRPperKW	float64
RRPperKW24hrs	float64
CustPricePerKW	float64
SupplyId	object
ExpenditureCode	object
PurchaseDate	datetime64[ns]

dtype: object

```
PurchaseWholesaleElec.head(2)
```

	Id	StationId	BasePrice	PeakPriceAdjustment	RRPperKW	RRPperKW24hrs	CustPricePerKW	SupplyId	ExpenditureCode
0	4926737	730023	0.55	0.0	0.01156	0.020110	0.570110	VIC	Wholesale_cos
1	3738844	730023	0.55	0.0	0.00895	-0.017512	0.532488	VIC	Wholesale_cos

Next steps: Generate code with PurchaseWholesaleElec | View recommended plots | New interactive sheet

```
# prompt: write the dataframe PurchaseWholesaleElec to a csv file

PurchaseWholesaleElec.to_csv('PurchaseWholesaleElec.csv', index=False)
```

Create UserPayment Database Table CSV

Table UserPayment {

Id int [PK, unique]

BookingDeposit money [ref: - UserPayment.TransAmount]

PurchaseId int

PaymentType varchar

CustPayAmount money [ref: - UserPayment.TransAmount]

TransAmount money

UserId int

StationId int

PaymentDate datetime

ChargingId int

OrderId int

}

```
df.head(2)
```

	sessionId	kwhTotal	CustPayAmount	created	ended	startTime	endTime	chargeTimeHrs	weekday	platform	distance
0	4926737	92.815080	52.914811	2023-10-07 07:18:43	2023-10-07 08:55:43	7	9	0 days 01:37:00	Saturday	android	NaN
1	3738844	80.584214	42.910162	2023-08-05 05:29:02	2023-08-05 06:53:02	5	7	0 days 01:24:00	Saturday	android	NaN

```
# prompt: add a column df[PaymentDate] = df[ended]

df['PaymentDate'] = df['ended']
```

```
# prompt: add a column df[TransAmount] = maximum of ( ( df[CustPayAmount] - df[BookingDeposit] ) or df[BookingDeposit] )

# Assuming df is already loaded and processed as in your provided code.

# Calculate TransAmount
df['TransAmount'] = np.maximum(df['CustPayAmount'] - df['BookingDeposit'], df['BookingDeposit'])
```

Need to Link Order, charging session and purchase ID together with the same Id

Generate | copy df[sessionId] to new column df[ChargingId] and df[OrderId] | Close

1 of 1 Use code with caution

```
# prompt: copy df[sessionId] to new column df[ChargingId] and df[OrderId]

# Assuming 'df' is your DataFrame and 'sessionId' column exists.
df['ChargingId'] = df['sessionId']
df['OrderId'] = df['sessionId']
```

```
# prompt: copy dataframe df to dataframe UserPayment

UserPayment = df.copy()
```

```
# prompt: change UserPayment[sessionId] to UserPayment[Id]

# Assuming df and PurchaseWholesaleElec are already defined DataFrames.

# Rename the 'sessionId' column to 'Id' in PurchaseWholesaleElec
if 'sessionId' in UserPayment.columns:
    UserPayment = UserPayment.rename(columns={'sessionId': 'Id'})
    print("Column 'sessionId' renamed to 'Id' in UserPayment")
else:
    print("Column 'sessionId' not found in UserPayment. No renaming performed.")
```

```
Column 'sessionId' renamed to 'Id' in UserPayment
```

```
UserPayment.head(2)
```

	Id	kwhTotal	CustPayAmount	created	ended	startTime	endTime	chargeTimeHrs	weekday	platform	distance	
0	4926737	92.815080	52.914811	2023-10-07 07:18:43	2023-10-07 08:55:43	7	9	0 days 01:37:00	Saturday	android	NaN	7
1	3738844	80.584214	42.910162	2023-08-05 05:29:02	2023-08-05 06:53:02	5	7	0 days 01:24:00	Saturday	android	NaN	7

```
# prompt: From dataframe UserPayment drop the following columns kwhTotal, created, ended, startTime, endTime, chargeTimeHrs,

columns_to_drop = ['kwhTotal', 'created', 'ended', 'startTime', 'endTime', 'chargeTimeHrs', 'weekday' , 'platform', 'distanc

UserPayment = UserPayment.drop(columns=columns_to_drop, errors='ignore')
```

```
# prompt: change UserPayment[stationId] to UserPayment[StationId], UserPayment[locationId] to UserPayment[LocationId],
# UserPayment[userId] to UserPayment[UserId]

# Rename the columns
UserPayment = UserPayment.rename(columns={'stationId': 'StationId', 'locationId': 'LocationId', 'userId': 'UserId'})
```

```
# prompt: copy UserPayment[Id] to UserPayment[PurchaseId]

# Assuming UserPayment DataFrame is already created as in the provided code.

UserPayment['PurchaseId'] = UserPayment['Id']
```

```
  # prompt: List the datatypes of UserPayment

  # Assuming 'UserPayment' DataFrame is already created as in the provided code.

  # Display the data types of columns in UserPayment
  UserPayment.dtypes
```

	0
Id	int64
CustPayAmount	float64
UserId	int64
StationId	int64
PaymentType	object
BookingDeposit	float64
PaymentDate	datetime64[ns]
TransAmount	float64
ChargingId	int64
OrderId	int64
PurchaseId	int64

dtype: object

```
# prompt: write the dataframe UserPayment to a csv file

UserPayment.to_csv('UserPayment.csv', index=False)
```

Create Stations Database Table CSV

Table Stations {

Id int [PK, unique]

ChargingPoints int

Latitude float

Longitude float

OperatorID varchar

Address varchar

Amenities varchar

BasePrice money [ref: - Stations.EstCost50KWChargerHr]

PeakPriceAdjustment money [ref: - Stations.EstCost50KWChargerHr]

EstCost50KWChargerHr float

SlotIDs int [unique]

ConnectionTypes varchar

PaymentTypes varchar

PowerOutput varchar

LocationName varchar

PostalCode varchar

LocationID varchar

}

```
# review the stations dataframe
df5.head(2)
```

	Station_no	Location Name	Latitude	Longitude	Town	Postal Code	City	Address	Plugs_Type2	Plugs_Three_Phase	Plugs_
2	2	Lonsdale St	-37.813437	144.955934	N.A	3000	Melbourne	535 Little Lonsdale St, Melbourne VIC 3004, Au...	0.0	0.0	
4	34	Secure Parking (450 Flinders Lane)	-37.818758	144.958866	N.A	3000	Melbourne	450 Flinders Ln, Melbourne VIC 3000, Australia	10.0	0.0	

```
# prompt: print the values of BasePrice  and PeakPriceAdjustment

print(f"BasePrice: {BasePrice}")
print(f"PeakPriceAdjustment: {PeakPriceAdjustment}")
```

```
BasePrice: 0.55
PeakPriceAdjustment: 0.0
```

```
# prompt: add a new column df5[BasePrice] = BasePrice and a new column df5[PeakPriceAdjustment] = PeakPriceAdjustment

# Add the new columns to df5 with default values that can be adjuected for individual stations later
df5['BasePrice'] = BasePrice
df5['PeakPriceAdjustment'] = PeakPriceAdjustment
```

```
# prompt: add a new column df5[EstCost50KWChargerHr] = 50* (df5['BasePrice']+df5['PeakPriceAdjustment'])

df5['EstCost50KWChargerHr'] = 50 * (df5['BasePrice'] + df5['PeakPriceAdjustment'])
```

```
# prompt: print payment_types

payment_types
```

```
['American Express', 'Visa', 'MasterCard', 'Cash']
```

```
# prompt: add a new column df5['PaymentTypes']  with each row having a default list of payment_types

# Assuming df5 is already defined as in the provided code.

# Add a new column 'PaymentTypes' with a default list of payment types for each row
df5['PaymentTypes'] = [payment_types for _ in range(len(df5))]
```

```
# review the stations dataframe
df5.head(5)
```